A MARSHALL CAVENDISH 26 COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE

UK £1.00
Republic of Ireland £1.25
Malta 85c
Australia $2.25
New Zealand $2.95



INPUT

Vol. 2 **No 26**

INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS
Front cover, Graeme Harris. Page 797, Stephen Smyth. Pages 798, 799, Diagram/MC. Pages 800, 801, Jeremy Gower/MC. Pages 804–809, Phil Dobson. Pages 815–817, Micky Finn. Page 819, Chris Lyon. Pages 820–821, Ian Stephen. Pages 224–228, Graeme Harris.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Printed by Artisan Presss, Leicester and Howard Hunt Litho, London.



HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland:
Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd,
Department 980, Newtown Road,
Hove, Sussex BN3 7DN

Australia: See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015

New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

Malta: Binders are available from local newsagents.

There are four binders each holding 13 issues.

BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services,
Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent.

COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:

Subscription Department, Marshall Cavendish Services Ltd,
Newtown Road, Hove, Sussex BN3 7DN

Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +
COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+
DRAGON 32 and 64
ZX81
VIC 20
TANDY TRS80 COLOUR COMPUTER

OUT OF THIS WORLD

- TARGETING
- RANGING
- ROUNDING THE EARTH
- MANOEUVRING IN SPACE
- PLANETARY MOTION

Reaching to the stars has long been one of Man's dreams. Following the article on objects in flight, this article shows how your micro can go some way to realizing that goal

In the article about trajectories on pages 740 to 747, you saw how the speed of a projectile can be split into a vertical and a horizontal component for analysis. You also saw how the range of a projectile can be varied by changing the angle of elevation and the speed at which it is shot. This article takes the study of moving bodies on to the next stage. It looks at the motion of objects in low gravity, and lets you examine their paths from distances near the Earth's surface to much farther out—in orbit.

Before launching into space, enter the first program, which demonstrates how a knowledge of trajectories can help you to make your own shooting games more interesting and challenging. If you use a Commodore 64, you need to make special arrangements to put the machine into a high-resolution graphics mode, for all but the first program. You can either use a Simons' BASIC cartridge or first enter the machine code hi-res utility on pages 748 to 751, and following articles. If you use the latter, you will need to prefix all the hi-res commands with an @. as explained in the first article. Vic users will need a Super Expander cartridge.



S

```
10 FOR n=Ø TO 31: READ a: POKE USR
   "a"+n,a: NEXT n
20 BORDER 4: PAPER Ø: INK 9: CLS
70 LET a=INT (RND*5): LET b=INT
   (RND*5)+26: LET c=INT (RND*8)+2:
   LET h=INT (RND*8)+2: LET st=INT
   (RND*1ØØ-c*8)+1: LET d=Ø
9Ø LET d=d+1
1ØØ CLS : GOSUB 3ØØ
11Ø INPUT "ANGLE?",a2
12Ø IF a2>89 OR a2<1 THEN GOTO 11Ø
13Ø INPUT "SPEED?",e
14Ø IF e=Ø THEN GOTO 1ØØ
16Ø LET an=a2*(PI/18Ø): LET x=Ø
17Ø LET x2=x+(a+1)*8
```



```
18Ø LET y=8+(x*TAN an-4*x*x/(e*e*COS
   an*COS an))
185 IF ATTR (21-INT (y/8),INT (x2/8))=6
   THEN GOTO 245
19Ø IF y<=Ø THEN GOTO 245
2ØØ IF (y>175 OR x2>255) AND d<1Ø
   THEN GOTO 9Ø
2Ø5 IF y>175 OR x2>255 THEN GOTO 27Ø
21Ø PLOT INK 8;x2,y: BEEP .Ø1,y/1Ø
22Ø LET x=x+3
23Ø GOTO 17Ø
245 IF x2>=b*8+3 AND x2<=b*8+1Ø
   THEN PRINT AT 21,b;CHR$ 145: FOR
   n=2Ø TO Ø STEP-1: BEEP .Ø1,n: NEXT
   n: GOTO 27Ø
246 IF d<1Ø THEN GOTO 9Ø
```

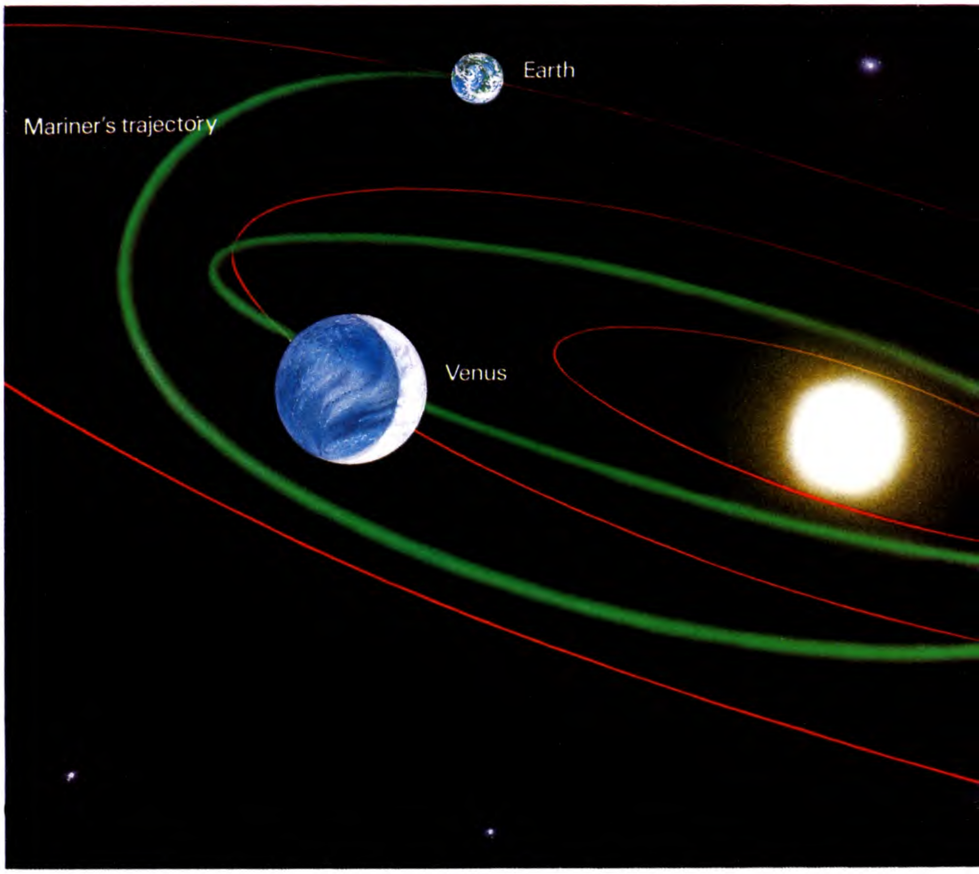


```
27Ø IF d=1Ø THEN PRINT AT
   8,1Ø;"USELESS!": GOTO 28Ø
275 PRINT INVERSE 1;AT 8,1Ø;"GOOD
   SHOT!";AT 1Ø,8;"YOU GOT IT IN□";d
28Ø PAUSE 1ØØ: PRINT BRIGHT 1; PAPER 2;
   INK 6;AT 13,8;"HAVE ANOTHER GO":
   PAUSE 2ØØ
29Ø GOTO 7Ø
3ØØ PRINT INK 5;AT 21,a;CHR$ 144;AT
   21,b;CHR$ 146
31Ø FOR n=1 TO c: PRINT AT 21-n+1,12;:
   FOR m=1 TO c: PRINT INK 6;CHR$ 147;:
   NEXT m: NEXT n
32Ø RETURN
5ØØ DATA 3,6,6Ø,4Ø,1Ø4,6Ø,126,255
51Ø DATA 36,9Ø,165,9Ø,6Ø,155,24,6Ø
52Ø DATA 24,36,66,153,153,66,36,127
53Ø DATA 28,42,85,17Ø,127,17Ø,85,255
```

C

```
99 GOSUB 13ØØØ
1ØØ A%=RND(1)*8: B%=RND(1)*8+31:
   C%=RND(1)*8+2: CT=Ø
11Ø CT=CT+1
115 PRINT "♥":SYS 832
12Ø GOSUB 9ØØØ
13Ø GOSUB 7ØØØ
14Ø GET I$: IF I$<>CHR$(13) THEN 14Ø
15Ø GOSUB 95ØØ
16Ø PRINT"♥"
17Ø INPUT "▤▟ANGLE";A2
18Ø IF A2<Ø OR A2>=9Ø THEN 17Ø
19Ø INPUT "▤▟▟▟SPEED";E
```

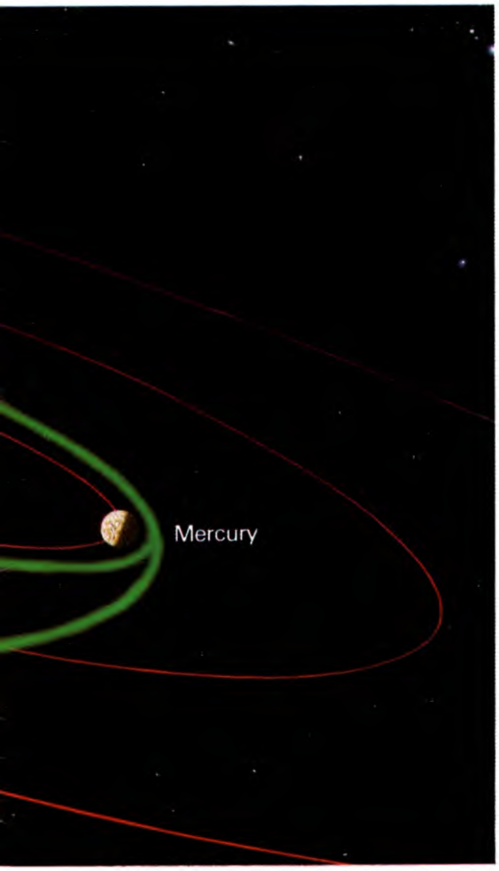
```
200 IF E=0 THEN 190
205 GOSUB 9000
210 AN=A2*(π/180):X3=8
220 X=X3+8*(A%+1)
230 H=8+X3*TAN(AN)-X3↑2/(E↑2*
    COS(AN)↑2)
240 Y=191-H: GOSUB 10000:
    FR=H/2+20: WF=33: GOSUB 11000
250 X3=X3+7
260 BY=24576+((Y-1) AND 248)*40+(X
    AND 504)+((Y-1) AND 7)
265 PO=(PEEK(BY)AND2↑(7-XAND7))
270 IF X<311 AND H>=8 AND H<183
    AND PO=0 THEN 220
273 HIT=ABS(X-(B%*8+4))<6 AND
    H<24
275 IF PO>0 OR HIT THEN FR=20:
    WF=129: GOSUB 11000
280 FOR D=1 TO 2000: NEXT
290 GOSUB 9500
310 IF CT<11 AND NOT HIT THEN 110
320 PRINT"♥"
322 FORD=1 TO 99:NEXT
325 GOSUB 9500
330 IF HIT THEN PRINT "GOOD
    SHOT!":PRINT:PRINT"GOT IT IN"CT
340 IF NOT HIT THEN PRINT "USELESS!"
350 PRINT:PRINT"HAVE ANOTHER GO"
360 FOR D=1 TO 4000: NEXT
370 GOTO 100
7000 V=1: FOR I=31936+A%*8 TO
    31936+A%*8+7: POKEI,V: V=V*2: NEXT
7010 FOR I=31936+B%*8 TO
    31936+B%*8+7: POKEI,255: NEXT
7020 FOR X=1 TO C%:FOR Y=1 TO
    C%:BA=32256+(13+X)*8-Y*320
7030 FOR I=BA TO BA+7:POKE I,63:NEXT
7040 NEXT:NEXT
7099 RETURN
9000 POKE 56576,150:POKE 53265,187:POKE
    53272,29:RETURN
9500 POKE 56576,151:POKE 53265,27:POKE
    53272,21:RETURN
10000 BY=24576+(YAND248)*40+
    (XAND504)+(YAND7):POKEBY,PEEK
    (BY)OR2↑(7-(XAND7))
10010 RETURN
11000 POKE 54296,10
11010 POKE 54278,251
11020 POKE 54276,WF
11030 POKE 54273,FR
11035 FOR D=1 TO 20:NEXT
11040 POKE 54276,WF-1
11050 RETURN
12000 DATA 169,0,133,251,169,96,133,
    252,169,0,168,145,251,200,208,251
12010 DATA 230,252,165,252,201,128,
    208,240
12020 DATA 162,0,169,7,157,0,68,157,
    0,69,157,0,70,157,232,70,232,208,
    241,96
```




```
13000 FOR Z=832 TO 875:READ X:POKE
    Z,X:NEXT Z: RETURN
```

```
70 A=INT(RND(1)*5):B=19-INT(RND
    (1)*5):C=INT(RND(1)*5)+2:H=INT
    (RND(1)*8)+2
75 SS=INT(RND(1)*100-C*8)+1:D=0:
    S=36877:POKES+1,15
90 D=D+1
100 GRAPHIC2:GOSUB 300:FOR Z=1 TO
    1000:NEXT Z
105 GRAPHIC 0
110 INPUT "♥ANGLE";A2
120 IF A2>89 OR A2<1 THEN 110
130 INPUT "♥SPEED";E
140 IF E=0 THEN 130
150 GRAPHIC 2:GOSUB 300
160 AN=A2*(π/180):X=0
170 X2=X+((A+1)*53)
180 Y=8+(X*TAN(AN)-4*X*X/
    (E*E*COS(AN)))
190 IF Y<=0 THEN 245
200 IF (Y>980 OR X2>1023) AND D<10
    THEN 90
205 IF Y>980 OR X2>1023 THEN 270
207 IF RDOT(X2,980-Y)=6 THEN 245
210 POINT 1,X2,980-Y:POKE S,
    128+(Y/10)
220 LET X=X+10:POKE S,0
230 GOTO 170
245 IF X2>=1023-(19-B+1)*51 AND
    X2=<1023-((19-B)*51) THEN 400
246 IF D<10 THEN 90
270 IF D=10 THEN:GRAPHIC 0:PRINT
    "USELESS!":GOTO 280
275 GRAPHIC 0:PRINT "GOOD SHOT!":PRINT
    "■YOU GOT IT IN";D
280 PRINT "■HAVE ANOTHER GO":FOR
    Z=1 TO 2000:NEXT Z
290 GOTO 70
300 GRAPHIC2:CHAR 19,A,"/":CHAR
    18,B,"▩":CHAR 19,B,"▀"
310 FOR N=1 TO C:FOR M=1 TO C:CHAR
    20-N,7+M,"●":NEXT M,N:RETURN
400 FOR Z=200 TO 127 STEP-1:POKE
    S,Z:NEXT Z:GOTO 270
```

```
10 VDU 23,225,3,6,60,40,104,60,126,255
20 VDU 23,226,36,90,165,90,60,155,24,60
30 VDU 23,227,24,36,66,153,153,66,36,127
40 VDU 23,228,28,42,85,170,127,170,85,255
50 MODE5:GCOL0,130:GCOL0,0:
    COLOUR130:COLOUR1
60 REPEAT
70 A=RND(4)-1:B=RND(4)+15:
    C=RND(8)+1:D=0
80 REPEAT
90 D=D+1
```




```
100 CLG:PROCSETUP
110 INPUTTAB(1,1)"ANGLE□",A2
120 IF A2>89 OR A2<1 THEN 1ØØ
130 INPUTTAB(1,2)"SPEED",E
140 IF E=Ø THEN 1ØØ
150 AN=RAD A2:X=8
160 REPEAT
170 X2=X+64*(A+1)
180 Y=64+X*TAN(AN)-X∧2/(E∧2*
    COS(AN)∧2)
190 PLOT69,X2,Y:SOUND1,-15,
    Y/5+2Ø,2
200 X=X+16
210 UNTILX2>128Ø OR Y<=48 OR ABS
    POINT(X2+9,Y)=1
220 IF Y<=48 THEN Y=32
230 IF X2>128Ø OR POINT (X2+
    8,Y)=-1 THEN X2=Ø:GOTO 25Ø
240 PRINTTAB((X2+16)/64,31-
    Y/32)CHR$(226);:SOUNDØ,
    -15,6,2Ø
250 FOR W=1 TO 4ØØØ:NEXT
260 CLG:UNTIL D=1Ø OR ABS
    (X2-B*64)<4Ø
270 IF D=1Ø THEN PRINTTAB(6,1Ø)
    "USELESS!"''''ELSE PRINTTAB
    (4,1Ø)"GOOD SHOT!"''''"□□YOU
    GOT IT IN□";D;"."
280 FOR W=1 TO 25ØØ:NEXT:
    PRINT''''"HAVE ANOTHER GO...":
    FOR W=1 TO 2ØØØ:NEXT:UNTIL Ø
290 DEF PROCSETUP:LOCAL D
300 PRINTTAB(A,3Ø)CHR$(225)
    TAB(B,3Ø)CHR$(227);
310 FOR D=1 TO C:PRINTTAB(6,3Ø-
    D+1) STRING$(C,CHR$(228));:NEXT
320 ENDPROC
```

```
10 PMODE4:DIM G(1),E(1),T(1),B(1)
20 FORK=1536TO2528 STEP32:
   READA:POKEK,A:NEXT
30 GET(Ø,Ø)-(7,7),G,G:
   GET(Ø,8)-(7,15),E,G
40 GET(Ø,16)-(7,23),T,G:
   GET(Ø,24)-(7,31),B,G
50 DATA 3,6,6Ø,4Ø,1Ø4,6Ø,126,255,36,
   9Ø,165,9Ø,6Ø,155,24,6Ø
60 DATA 24,36,66,153,153,66,36,127,
   28,42,85,17Ø,127,17Ø,85,255
70 A=RND(51)-1:B=RND(51)+197:
   C=RND(8):H=RND(8)+1:ST=RND
   (1ØØ-C*8)+7Ø:D=Ø:CLS
80 D=D+1
90 PCLS:SCREEN1,1:GOSUB32Ø
100 IF INKEY$="" THEN 1ØØ
110 PRINT:INPUT" ANGLE□";A2
120 IF A2>89 OR A2<1 THEN 11Ø
130 INPUT"□SPEED□";E
140 IF E=Ø THEN 13Ø
150 SCREEN1,1
160 AN=A2*ATN(1)/45:X=Ø
170 X2=X+A+8
180 Y=183-(X*TAN(AN)-4*X*X/
    (E*E*COS(AN)*COS(AN)))
190 IF Y>19Ø THEN 25Ø
200 IF X2>=ST AND X2<ST+C*8+7
    AND Y>183-H*8 THEN 25Ø
210 IF Y>Ø THENPSET(X2,Y,5):
    SOUND2ØØ-Y,1 ELSESOUND255
    AND(2ØØ-Y),1
220 X=X+3
230 IF X2<255 THEN 17Ø
240 GOTO29Ø
250 IF Y>19Ø THENY=184
260 PUT(X2-4,Y)-(X2+3,Y+7),
    E,PSET:PLAY"T5ØO1ADECBFGAEDBGDE"
270 PUT(A,184)-(A+7,191),G,
    PSET:PUT(B,184)-(B+7,191),
    T,PSET
280 IF X2>=B AND X2<B+7 THEN 3ØØ
290 IF D<1Ø THEN 8Ø
300 CLS:IFD=1Ø THENPRINT@41,
    "USELESS !" ELSEPRINT@41,"GOOD
    SHOT !":PRINT@164,"YOU GOT
    IT IN";D;"SHOTS."
310 FORW=1TO3ØØØ:NEXT:PRINT:
    PRINT"□□HAVE ANOTHER GO...":
    FORW=1TO12ØØ:NEXT:GOTO7Ø
320 FORX=ØTOC:FORY=ØTOH:PUT
    (ST+X*8,184-8*Y)-(ST+X*8+7,
    191-8*Y),B,PSET
330 NEXTY,X
340 PUT(A,184)-(A+7,191),G,
    PSET:PUT(B,184)-(B+7,191),
    T,PSET:RETURN
```

The program prompts you to enter take-off speed and angle of elevation to get a shot to travel from a point near the bottom left of the screen to a target near the bottom right. The game is made difficult by setting the firing point and the target at random distances apart in each series of attempts—and, even more importantly, by having a barrier of random size at a random point between starting and finishing points. Whatever trajectory you pick must be high enough to lob the shot over this barrier.

RANGING

This program gives a good example of how well the brain makes judgements. Merely by looking at the positions of the gun, obstacle and target, you can estimate the speed and angle required to produce a curve that sails the shot neatly over the obstacle to hit the target. With a little practice, you will find it possible to score a surprisingly high proportion of direct hits.

But ranging like this is actually a bit hit-and-miss, especially when you do not have a clear side view. What is far more likely to happen is that you know the approximate distance of the target, and have to calculate the angle and speed required. By seeing whether the shot is under or over the required length, you can make progressively more accurate calculations.

The next program shows how this is done. Clear the first program (after SAVEing it if you want to re-use the game), and enter these lines. (Commodore 64 users should remember to fit a Simons' BASIC cartridge or enter their machine code routine first.)

```
10 CLS
20 INPUT "FIRING SPEED (1-1ØØØØ m/s)",sp
30 IF sp<1 OR sp>1ØØØØ THEN GOTO 2Ø
40 INPUT AT 4,Ø;"FIRING ANGLE (1-9Ø
   DEG)",a
50 IF a<1 OR a>=9Ø THEN GOTO 4Ø
60 LET a=a*(PI/18Ø)
70 LET r=sp*sp*SIN (2*a)/1Ø
80 PRINT AT 1Ø,3;"THE RANGE IS□";INT
   (r+.5);"□metres"
90 PRINT AT 21,1;"ANY KEY FOR ANOTHER
   GO (Ø END)"
100 PAUSE Ø: LET a$=INKEY$: IF a$=""
    THEN GOTO 1ØØ
110 IF a$<>"Ø" THEN GOTO 1Ø
```

```
20 PRINT "♥FIRING SPEED (1-1ØØØØ
   M/S)":INPUT SP
3Ø IF SP<1 OR SP>1ØØØØ THEN 2Ø
4Ø PRINT "♥FIRING ANGLE(1-9Ø
   DEG)":INPUT A
5Ø IF A<1 OR A>89 THEN 4Ø
6Ø A=A*(π/18Ø)
7Ø R=SP*SP*SIN(2*A)/1Ø
8Ø PRINT "♥THE RANGE IS";INT(R+.5);
   "METRES"
9Ø PRINT "▣HIT ANY KEY FOR ANOTHER
   GO (Ø TO END)"
1ØØ GET D$:IF D$="" THEN 1ØØ
11Ø IF D$<>"Ø" THEN RUN
```

```
1Ø MODE1
2Ø INPUT TAB(5,2)"FIRING SPEED (1-1ØØØØ
   m/s)□",SP
3Ø IF SP<1 OR SP>1ØØØØ THEN GOTO 2Ø
4Ø INPUT TAB(5,4)"FIRING ANGLE (1-9Ø
   DEG)□",A
5Ø IF A<1 OR A>9Ø THEN PRINT
   TAB(3Ø,4)"□□□□□":GOTO 4Ø
6Ø A=RAD A
7Ø R=SP*SP*SIN(2*A)/1Ø
8Ø PRINT TAB(8,15)"THE RANGE
   IS□";INT(R+.5);"□metres"
9Ø PRINT TAB(4,3Ø)"ANY KEY FOR ANOTHER
   GO (Ø TO END)"
1ØØ D=GET
11Ø IF D=48 THEN END ELSE RUN
```

```
1Ø CLS
2Ø INPUT"□FIRING SPEED (1-1ØØØØ
   M/S)□□□";SP
3Ø IF SP<1 OR SP>1ØØØØ THEN 1Ø
4Ø INPUT"□FIRING ANGLE (1-9Ø
   DEG)□";A
5Ø IF A<1 OR A>=9Ø THEN 4Ø
6Ø A=A*ATN(1)/45
7Ø R=SP*SP*SIN(2*A)/1Ø
8Ø PRINT:PRINT"□THE RANGE
   IS";INT(R+.5);"METRES"
9Ø PRINT:PRINT"□ANY KEY FOR ANOTHER
   GO□□□□□□□□□□□□
   (Ø TO END)"
1ØØ A$=INKEY$:IF A$="" THEN 1ØØ
11Ø IF A$="Ø" THEN CLS:END ELSE 1Ø
```

This program lets you input values for initial speed (Line 2Ø) and angle of elevation (Line 4Ø). Line 7Ø then calculates and displays the range that these values would give to a projectile. The variable R is the range, SP is the speed, A is the angle, and the 10 is the approximate value of gravity near the Earth's surface (9.81 is the actual figure).

The effect of friction in the air is ignored, but in real life trials it must be accounted for. The highest firing speed of a cannon is more than 2000 m/s. This has been fired straight up to send objects high above the Earth for research. The objects reached as high as 180 km, but without air friction the height would have been as much as 250 km.

ROUNDING THE EARTH

If the object is fired at an angle that takes it high above the Earth, the effect of friction is less, but if you wish it to have very long range, you must take into account the curvature of the Earth. One of the first people to consider the longest possible range of a projectile over the Earth's surface was the British scientist, Sir Isaac Newton.

Newton imagined a powerful gun at the top of a mountain high enough to be out of the Earth's atmosphere. Shots fired at increasing muzzle speed would travel increasingly farther on a flat Earth. But because the Earth curves, the surface falls away from the shot, which can travel to an even greater distance than if the Earth were flat.

Eventually, Newton argued, a shot would start so fast that it would never hit the ground, but should, in theory, hit him in the back of the head. As fast as the cannonball fell to the ground, the ground would 'fall' away beneath it, so the shot would remain in 'free fall' for ever—it would orbit the Earth.

Once an object has escaped from a planet's gravity, its speed and distance from the planet determine the type of orbit it follows—circular or elliptical.

To be able to make predictions and measurements about a planet or satellite, its orbit must be known precisely—we must know the exact shape of the ellipse. The degree of 'squashedness' of an ellipse is its eccentricity (E). This is the proportion of the length of the ellipse to its width. If E equals one, the ellipse is as wide as it is long—it is a circle.

Enter and RUN the next program to see the effect of varying E between values less than one and values greater than one:

```
1Ø CLS
3Ø INPUT "ECCENTRICITY (Ø.1-1.9)",e
4Ø IF e<.1 OR e>1.9 THEN GOTO 3Ø
5Ø PLOT 127,87+e*4Ø
6Ø FOR a=Ø TO 2*PI+.2 STEP .1
7Ø DRAW 127+(4Ø*SIN a)-PEEK
   23677,87+(e*4Ø*COS a)-PEEK 23678
8Ø NEXT a
9Ø PRINT AT 21,1;"ANY KEY FOR ANOTHER
   GO (Ø END)"
1ØØ PAUSE Ø: LET a$=INKEY$: IF a$=""
   THEN GOTO 1ØØ
11Ø IF a$<>"Ø" THEN GOTO 1Ø
```

```
3Ø PRINT "♥ECCENTRICITY
   (Ø.1-1.9)":INPUT E
4Ø IF E<.1 OR E>1.9 THEN 3Ø
5Ø HIRES Ø,1:X=16Ø:Y=1ØØ-E*5Ø
6Ø FOR A=Ø TO 2*π+.2 STEP.1
7Ø LINE X,Y,16Ø+8Ø*SIN(A),
   1ØØ-(E*5Ø*COS(A)),1
8Ø X=16Ø+8Ø*SIN(A):Y=
   1ØØ-(E*5Ø*COS(A)):NEXT A
9Ø FOR Z=1 TO 2ØØØ:NEXT Z:NRM:RUN
```

```
3Ø PRINT"♥ECCENTRICITY":PRINT
   "(Ø.1-1.9)":INPUT E
4Ø IF E<.1 OR E>1.9 THEN 3Ø
5Ø GRAPHIC 2:X=512:Y=512-E*25Ø:
   POINT 1,X,Y
6Ø FOR A=Ø TO 2*π+.2 STEP.1
7Ø DRAW 1 TO 512+25Ø*SIN(A),
   512-(E*25Ø*COS(A)):NEXT A
9Ø FOR Z=1 TO 2ØØØ:NEXT Z:GRAPHIC
   Ø:RUN
```



```
1Ø MODE1
2Ø VDU29,65Ø;5ØØ;
3Ø INPUT TAB(Ø,2)"ECCENTRICITY"
   "(Ø.1-1.9)□",E
4Ø IF E<Ø.1 OR E>1.9 THEN RUN
5Ø MOVE Ø,E*25Ø
6Ø FOR A=Ø TO 2*PI+Ø.2 STEP Ø.1
7Ø DRAW 25Ø*SIN A,E*25Ø*COS A
8Ø NEXT A
9Ø PRINT TAB(4,3Ø)"ANY KEY FOR ANOTHER
   GO (Ø TO END)"
1ØØ D=GET
11Ø IF D=48 THEN END ELSE RUN
```

```
1Ø PMODE4,1:PCLS
3Ø CLS:INPUT"ECCENTRICITY (.1 TO 1.9)□";E
4Ø IF E<.1 OR E>1.9 THEN 3Ø
5Ø SCREEN1,1
7Ø CIRCLE(128,96),48,5,E
1ØØ A$=INKEY$:IF A$="" THEN 1ØØ
11Ø IF A$="Ø" THEN CLS:END ELSE 3Ø
```

This program loops between Lines 3Ø and 11Ø to draw ellipses for which you INPUT the value of E at Line 3Ø. The Dragon and Tandy programs use a CIRCLE command (Line 7Ø) to draw the curves, but the others step through a FOR . . . NEXT loop (Lines 6Ø to 8Ø) and DRAW between each point.

Enter values of E in the range shown on the screen and verify that E=1 gives a circle, E less than 1 gives downwards squashed ellipses and E greater than 1 gives sideways squashed ellipses. (On the Commodores E=1.5 looks more like a circle). So any orbit can be considered as an ellipse with suitable E value.

MANOEUVRING IN SPACE

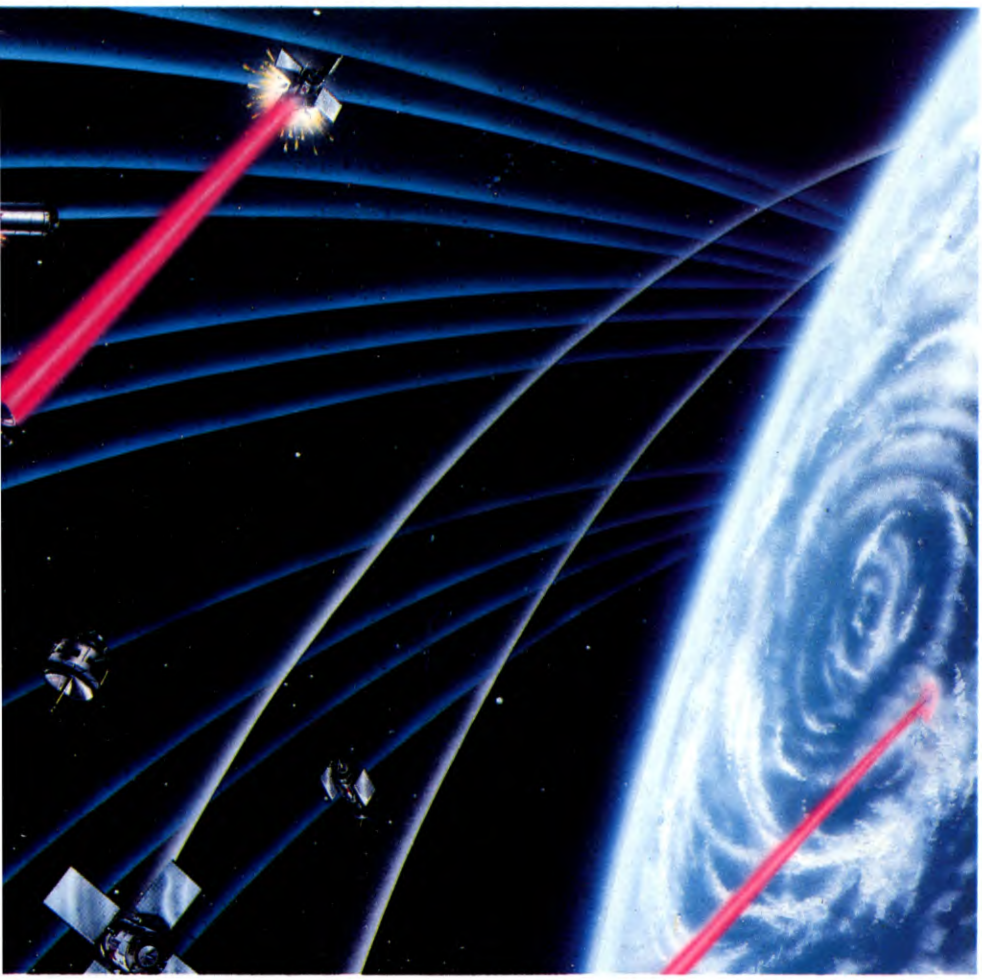
Once in orbit, a satellite or spacecraft needs no power to keep it there, because it is falling freely. Any use of the rockets will move the craft out to a higher orbit or in to a lower one.

The smaller the radius of the orbit, the faster the object must move. Enter the next program to see how this works:

```
1Ø CLS : LET gc=Ø
4Ø LET r=4Ø: LET rt=1Ø+INT (RND*6Ø): IF
   ABS (r-rt)<1Ø THEN GOTO 4Ø
5Ø CIRCLE 127,87,4
6Ø LET s=.1: LET a=Ø: LET at=INT
   (RND*1Ø)+1: LET f=Ø
8Ø LET a=a+s
1ØØ LET at=at+.1*SQR ((4Ø/rt)↑3)
11Ø IF INKEY$="7" AND r<85 THEN LET
   f=f+1: LET r=r+2: LET s=s*SQR
   (((r-2)/R)↑3: GOTO 13Ø
115 IF INKEY$="6" AND r>8 THEN LET
   f=f+1: LET r=r-2: LET s=s*SQR
   (((r+2)/r)↑3): GOTO 13Ø
12Ø FOR p=1 TO 5: NEXT p
13Ø LET x=r*SIN a: LET y=r*COS a
14Ø LET xt=rt*SIN at: LET yt=rt*COS at
15Ø PLOT 127+x,87+y
16Ø BEEP .Ø2,r/2
165 IF gc<>Ø THEN PLOT OVER
   1;127+ox,87+oy
17Ø PLOT OVER 1;127+xt,87+yt: LET
   ox=xt: LET oy=yt: LET gc=1
18Ø IF ABS (x-xt)>4 OR ABS (y-yt)>4
   THEN GOTO 8Ø
19Ø PRINT AT 2Ø,12;f;"□BURNS"
2ØØ GOTO 2ØØ
```

```
1Ø HIRES Ø,1
2Ø POKE 54296,15:POKE 54277,64
4Ø R=3Ø:RT=INT(RND(1)*7Ø)+1Ø:IF
   ABS(R-RT)<2Ø THEN 4Ø
5Ø TEXT 157,97,"*",1,1,1
6Ø S=.1:A=Ø:AT=INT(RND(1)*1Ø)+1:F=Ø
8Ø A=A+S
95 LT=AT
1ØØ AT=AT+.1*SQR((4Ø/RT)↑3)
11Ø GET A$:IF A$="■" AND
   R<95 THEN F=F+1:R=R+1:S=S*
   SQR(((R-1)/R)↑3)
12Ø IF A$="■" AND R>8 THEN
   F=F+1:R=R-1:S=S*SQR(((R+1)/R)
   ↑3)
13Ø X=R*SIN(A):Y=R*COS(A)
14Ø XT=RT*SIN(AT):YT=RT*COS(AT)
15Ø PLOT 16Ø+X,1ØØ-Y,1
16Ø POKE 54276,17:POKE 54273,
   1+R:FOR Z=1 TO 5:NEXT Z
165 POKE 54276,Ø:POKE 54273,Ø
17Ø PLOT 16Ø+XT,1ØØ-YT,1
175 PLOT 16Ø+RT*SIN(LT),
   1ØØ-RT*COS(LT),Ø
18Ø IF ABS(X-XT)>3 OR ABS
   (Y-YT)>3 THEN 8Ø
19Ø NRM:PRINT "■YOU USED";F;"BURNS"
```

```
1Ø GRAPHIC 2
2Ø POKE 36878,15
4Ø R=2ØØ:RT=INT(RND(1)*3ØØ)+5Ø:IF
   ABS(R-RT)<1ØØ THEN 4Ø
5Ø CHAR 9,9,"*"
6Ø S=.1:A=Ø:AT=INT(RND(1)*1Ø)+1:
   F=Ø
```



```
80 A=A+S
95 LT=AT
100 AT=AT+.1*SQR((200/RT)↑3)
110 GET A$:IF A$="■" AND R<480
    THENF=F+1:R=R+4:S=S*SQR
    (((R-4)/R)↑3)
120 IF A$="■" AND R>8 THENF=
    F+1:R=R-4:S=S*SQR(((R+4)/
    R+4)/R)↑3)
130 X=R*SIN(A):Y=R*COS(A)
140 XT=RT*SIN(AT):YT=RT*COS(AT)
150 POINT 1,490+X,490-Y
160 POKE 36876,128+(R/90):FOR Z=1 TO
    5:NEXT Z:POKE 36876,Ø
170 POINT 1,490+XT,490-YT
175 POINT Ø,490+RT*SIN(LT),
    490-RT*COS(LT)
180 IF ABS(X-XT)>20 OR
    ABS(Y-YT)>20 THEN 80
190 GRAPHIC Ø:PRINT "■YOU
    USED";F;"BURNS"
```

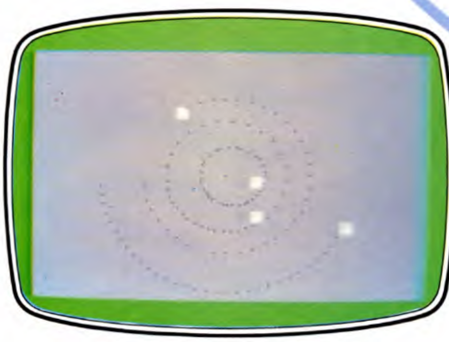
```
10 MODE1:VDU5
20 VDU29,640;512;
30 *FX4,1
40 REPEAT:R=200:RT=50+RND(350):
   UNTIL ABS(R-RT)>100
50 MOVE -16,12:GCOLØ,2:PRINT"*":
   GCOLØ,3
60 S=Ø.1:A=Ø:AT=RND(1Ø):F=Ø
70 REPEAT
80 A=A+S
90 GCOLØ,Ø:PLOT69,RT*SIN AT,RT*COS AT
100 AT=AT+Ø.1*SQR((200/RT)∧3)
110 IF INKEY(-58) AND R<500 THEN
    F=F+1:R=R+2:S=S*SQR
    (((R-2)/R)∧3) ELSE IF INKEY(-42)
    AND R>40 THEN
    F=F+1:R=R-2:S=S*
    SQR(((R+2)/R)∧3) ELSE FOR P=1 TO
    5Ø:NEXT P
120 GCOLØ,3
130 X=R*SIN A:Y=R*COS A
140 XT=RT*SIN AT:YT=RT*COS AT
150 GCOLØ,3:PLOT69,X,Y
160 SOUND 1,-5,R/2,1:GCOL Ø,1
170 GCOLØ,1:PLOT69,XT,YT
180 UNTIL ABS(X-XT)<20 AND
    ABS(Y-YT)<20
190 GCOLØ,3:VDU4:PRINTTAB(Ø,1)F;
    "□BURNS"
200 GOTO 200
```

On the Tandy, change the 223s in Line 11Ø to 247.

```
10 PMODE4:PCLS:SCREEN1,1
40 R=3Ø:RT=1Ø+RND(7Ø):IF
   ABS(R-RT)<2Ø THEN 4Ø
```



A simple game using trajectories



The orbits of the four innermost planets

```
50 DRAW"BM128,96S8NUNRNDL"
60 S=.1:A=Ø:AT=RND(1Ø):F=Ø
80 A=A+S
95 LT=AT
100 AT=AT+.1*SQR((4Ø/RT)∧3)
110 IF PEEK(341)=223 AND R<95 THEN
    F=F+1:R=R+1:S=S*SQR
    (((R-1)/R)∧3) ELSE IF PEEK(342)
    =223 AND R>8 THEN
    F=F+1:R=R-1:
    S=S*SQR(((R+1)/R)∧3) ELSE
    FORK=1TO25:NEXT
130 X=R*SIN(A):Y=R*COS(A)
140 XT=RT*SIN(AT):YT=RT*COS(AT)
150 PSET(128+X,96-Y,5)
160 SOUNDR*3,1
170 PSET(128+XT,96-YT,5)
175 PSET(128+RT*SIN(LT),96-RT*
    COS(LT),Ø)
180 IF ABS(X-XT)>=4 OR
    ABS(Y-YT)>=4 THEN 8Ø
190 CLS:PRINT"□YOU USED";F; "BURNS"
```

Run the program to see the trail of an orbiting craft and a target satellite without a trail. Try to match the orbit of the craft with that of the satellite, using the up and down arrow keys (6 and 7 on the Spectrum). Line 4Ø sets the radius R of the craft's orbit as 200 and the radius of the target as a random value. Line 6Ø sets variables for the starting positions.

The crux of the program lies at Line 1ØØ, which makes use of another important physical law: the square of the time to make one orbit, divided by the cube of the radius, is constant. This is the reason for the SQR and power of 3 in this line.

To manoeuvre, use the up and down arrow keys to increase or decrease the size of the orbit. Remember that you go faster when the orbit is smaller.

PLANETARY MOTION

In reality, manoeuvring in and out of orbit is much harder than the last program suggests. It is easier to transfer from elliptical orbits, but the vastness of space makes it difficult to locate a target. Then there is the effect of gravity to complicate matters. The gravity of the Sun, Moon and planets all have an effect on the motion of a spacecraft.

In practice, powerful computers are the tools of space navigators. They control the speed, time and direction of rocket burns to maintain a required path.

Once in the correct path, a craft falls freely in the solar system's gravitational field. It needs few course corrections during the months and years of travel—just as the planets lap the Sun predictably year after year. The next program lets you view the planets in motion:

```
10 BORDER 4: CLS : LET gc=Ø
20 DIM d(9): DIM p(9): DIM i(9): DIM a(9):
   DIM b(9): DIM x(9): DIM y(9)
30 FOR t=1 TO 9: READ d(t),p(t): NEXT t
40 INPUT "How many planets (1-9)",s
50 IF s<1 OR s>9 THEN GOTO 4Ø
60 LET sc=d(s)/325: LET t=p(s)/75
70 PRINT "There is a□";INT t;"□day
   delay""between each point": PRINT
   #1;"PRESS SPACE TO CONTINUE"
75 IF INKEY$<>CHR$ 32 THEN GOTO 75
80 CLS
100 PRINT #1;"PRESS SPACE TO HAVE
    ANOTHER GO"
110 LET n=1: LET m=1: LET g=PI/18Ø
120 FOR q=1 TO s
130 LET r=d(q)/sc: LET a=r: LET b=r: LET
    e=Ø: LET p=p(q)/t
140 IF q=1 THEN LET e=.2: LET b=a*.98
150 IF q=8 THEN LET e=.26: LET b=a*.96
160 LET i(q)=i(q)+360/p
180 LET y=g*i(q): LET x=INT (a*(COS
    y-e)): LET y=INT (b*SIN y)
200 IF gc<>Ø THEN PLOT BRIGHT
    Ø;127+x(q),87+y(q)
210 PLOT BRIGHT 1;127+a(q)/4,87
    +b(q)/4: LET x(q)=a(q)/4: LET
    y(q)=b(q)/4
240 LET a(q)=x: LET b(q)=y: NEXT q
```

```
250 LET m=m+t
260 IF INKEY$=CHR$ 32 THEN RUN
270 LET gc=1: GOTO 120
280 DATA 58,88,108,225,150,365,228,
    687
290 DATA 778,4333,1427,10759,2870,
    30685
300 DATA 4497,60190,5969,90741
```

```
10 PRINT "■■":NRM:COLOUR 0,1
20 DIM D(8),P(8),I(8),A(8),B(8)
30 FOR T=0 TO 8:READ D(T),P(T):
   NEXT T
40 PRINT "■HOW MANY PLANETS
   (1-9)":INPUT S
50 IF S<1 OR S>9 THEN 40
60 S=S-1:SC=D(S)/90:T=P(S)/75
70 PRINT "■THERE IS A";INT(T);
   "DAY":PRINT "DELAY BETWEEN EACH
   POINT"
75 PRINT "■HIT SPACE TO CONTINUE"
76 GET A$:IF A$<>"□" THEN 76
80 HIRES 1,0
110 G=π/180
120 FOR Q=0 TO S
130 R=D(Q)/SC:A=R:B=R:E=0:
    P=P(Q)/T
140 IF Q=0 THEN E=.2:B=A*.98
150 IF Q=8 THEN E=.26:B=A*.96
155 IF S>3 AND Q<4 THEN 245
160 IF P>3 THEN P=INT(P+.5)
170 I(Q)=I(Q)+360/P
180 Y=G*I(Q):X=INT(A*(COS(Y)-E)):
    Y=INT(B*SIN(Y))
200 TEXT 157+A(Q),97-B(Q),"○",0,
    1,1
220 PLOT 160+A(Q),100-B(Q),1
225 TEXT 157,97,"●",1,1,1
230 TEXT 157+X,97-Y,"○",1,1,1
240 A(Q)=X:B(Q)=Y
245 NEXT Q
250 M=M+T
260 GET A$:IF A$="□" THEN RUN
270 GOTO 120
280 DATA 58,88,108,225,150,365,228, 687
290 DATA 778,4333,1427,10759,2870, 30685
300 DATA 4497,60190,5969,90741
```

```
10 GRAPHIC 0:PRINT "■":COLOUR 0,0,1,1
20 DIM D(8),P(8),I(8),A(8),B(8)
30 FOR T=0 TO 8:READ D(T),P(T):NEXT T
40 PRINT "■HOW MANY PLANETS
   (1-9)":INPUT S
50 IF S<1 OR S>9 THEN 40
60 S=S-1:SC=D(S)/325:T=P(S)/75
70 PRINT "■THERE IS A";INT(T);
   "DAY":PRINT "DELAY BETWEEN
   EACH □□□□POINT"
75 PRINT "■HIT SPACE TO CONTINUE"
76 GET A$:IF A$<>"□" THEN 76
80 GRAPHIC 2
90 CHAR 9,9,"●"
110 G=π/180
120 FOR Q=0 TO S
130 R=D(Q)/SC:A=R:B=R:E=0:
    P=P(Q)/T
140 IF Q=0 THEN E=.2:B=A*.98
150 IF Q=8 THEN E=.26:B=A*.96
155 IF S>3 AND Q<4 THEN 245
160 IF P>3 THEN P=INT(P+.5)
170 I(Q)=I(Q)+360/P
180 Y=G*I(Q):X=INT(A*(COS(Y)-E)):
    Y=INT(B*SIN(Y))
230 POINT 1,500+X,500-Y
245 NEXT Q
250 M=M+T
260 GET A$:IF A$="□" THEN RUN
270 GOTO 120
280 DATA 58,88,108,225,150,365,228,
    687
290 DATA 778,4333,1427,10759,2870,
    30685
300 DATA 4497,60190,5969,90741
```

```
10 MODE1
20 DIM D(8),P%(8),I(8),A%(8),B%(8)
30 FOR T=0 TO 8:READ D(T),P%(T):
   NEXT
40 INPUT TAB(0,1)"HOW MANY PLANETS
   (1-9)□",S%
50 IF S%<1 OR S%>9 THEN CLS:GOTO 40
60 S%=S%-1:SC=D(S%)/325:T=P%(S%)
   /75
70 PRINT"THERE IS A□";INT
   T"□DAY"'"DELAY BETWEEN EACH
   POINT"'"PRESS SPACE TO
   CONTINUE":REPEAT UNTIL GET=32
80 MODE1
90 VDU 29,640;512;
100 PRINT TAB(3,31)"PRESS SPACE TO HAVE
    ANOTHER GO"
110 N%=0:M=0:G=PI/180:VDU5
120 FOR Q=0 TO S%
130 R=D(Q)/SC:A=R:B=R:E=0:
    P=P%(Q)/T
140 IF Q=0 THEN E=0.2: B=A*0.98
150 IF Q=8 THEN E=0.26: B=A*0.98
160 IF P>3 THEN P=INT(P+0.5)
170 I(Q)=I(Q)+360/P
180 Y=G*I(Q):X=INT(A*(COS(Y)-E)):
    Y=INT(B*SIN(Y))
190 GCOL0,0
200 MOVE A%(Q)-12,B%(Q)+24:PRINT"."
210 GCOL0,3
220 PLOT 69,A%(Q),B%(Q)
230 MOVE X-12,Y+24:PRINT"."
240 A%(Q)=X:B%(Q)=Y:NEXT
250 M=M+T
260 IF INKEY(-99) THEN RUN
270 GOTO 120
280 DATA 58,88,108,225,150,365,228,
    687
290 DATA 778,4333,1427,10759,2870,
    30685
300 DATA 4497,60190,5969,90741
```

```
10 PMODE4:PCLS
20 DIM D(8),P(8),I(8),A(8),B(8)
30 FOR T=0 TO 8:READ D(T),P(T):NEXT
40 CLS:INPUT"HOW MANY PLANETS
   (1-9)□";S
50 IF S<1 OR S>9 THEN 40
60 S=S-1:SC=D(S)/90:T=P(S)/75
70 PRINT:PRINT"THERE IS A";INT(T);
   "DAY":PRINT"DELAY BETWEEN EACH
   POINT":PRINT:PRINT:PRINT"PRESS
   SPACE TO CONTINUE"
75 IF INKEY$<>"□" THEN 75
80 SCREEN1,1
90 CIRCLE(128,96),1,5
110 G=ATN(1)/45
120 FORQ=0 TO S
130 R=D(Q)/SC:A=R:B=R:E=0:
    P=P(Q)/T
140 IF Q=0 THEN E=.2:B=A*.98
150 IF Q=8 THEN E=.26:B=A*.96
155 IF S>5 AND Q<6 THEN 245
160 IF P>3 THEN P=INT(P+.5)
170 I(Q)=I(Q)+360/P
180 Y=G*I(Q):X=INT(A*(COS(Y)-E)):
    Y=INT(B*SIN(Y))
200 CIRCLE(128+A(Q),96-B(Q)),1,0
220 PSET (128+A(Q),96-B(Q),5)
230 CIRCLE(128+X,96-Y),1,5
240 A(Q)=X:B(Q)=Y
245 NEXT
250 M=M+T
260 IF INKEY$="□" THEN RUN
270 GOTO 120
280 DATA 58,88,108,225,150,365,228,
    687
290 DATA 778,4333,1427,10759,2870,
    30685
300 DATA 4497,60190,5969,90741
```

When you RUN this program, you are prompted to enter a value to select how many planets you wish to view. The larger the number (between 1 and 9), the more complicated the picture. To find out which planets you are viewing, remember that Mercury is nearest the Sun, then come Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune and Pluto. RUN the program with different input values and notice that the orbits of two planets—Mercury and Pluto—are distinctly elliptical orbits. In fact, they are all elliptical, only some have so little eccentricity that they approximate closely to circles.

SNAKES AND ADDERS

Guide the starving snake to the food in *INPUT*'s snake game. Gobbling nourishing numbers will make the baby snake grow into a huge serpent if you're skilful

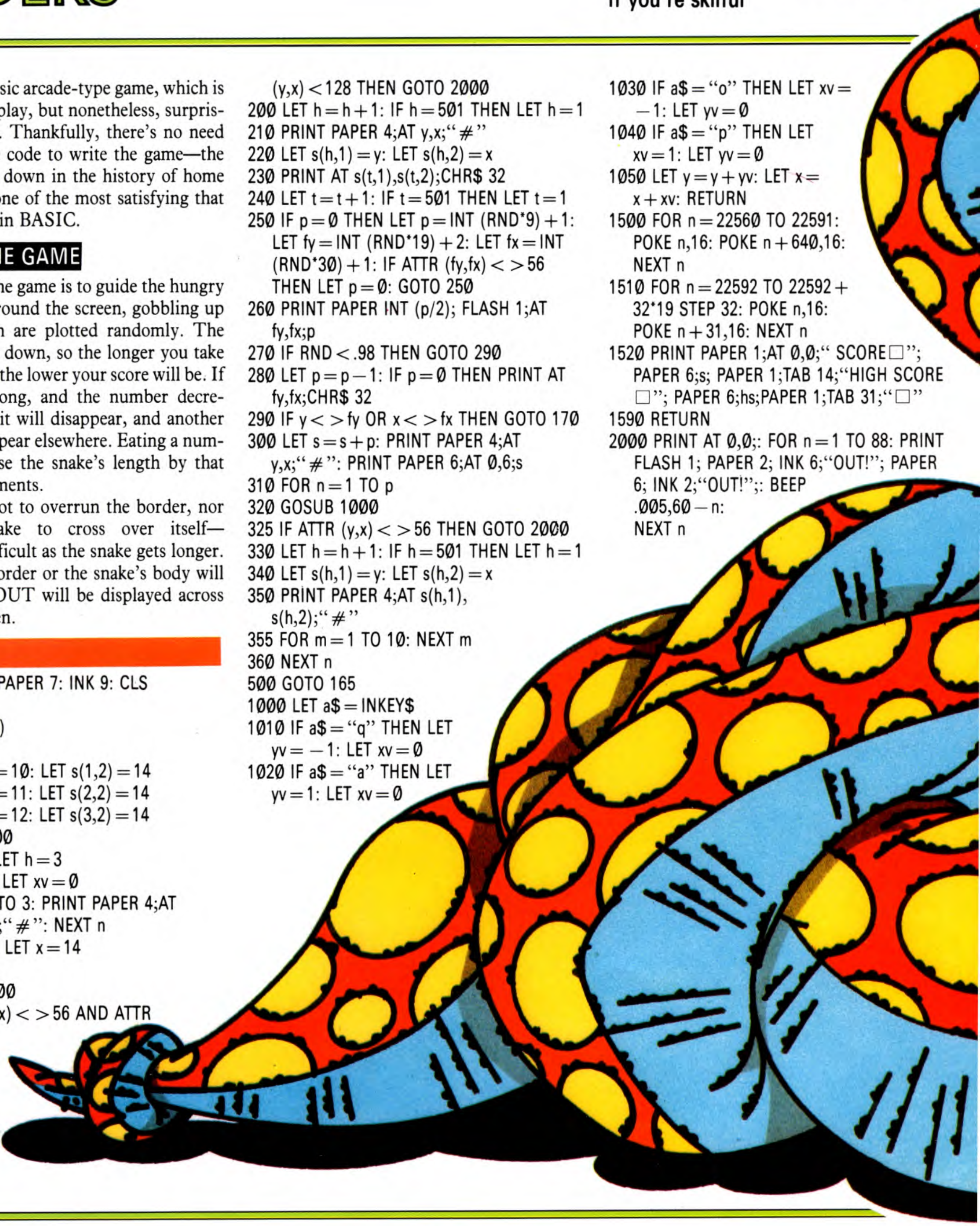
'Snake' is a classic arcade-type game, which is very simple to play, but nonetheless, surprisingly addictive. Thankfully, there's no need to use machine code to write the game—the game has gone down in the history of home computing as one of the most satisfying that can be written in BASIC.

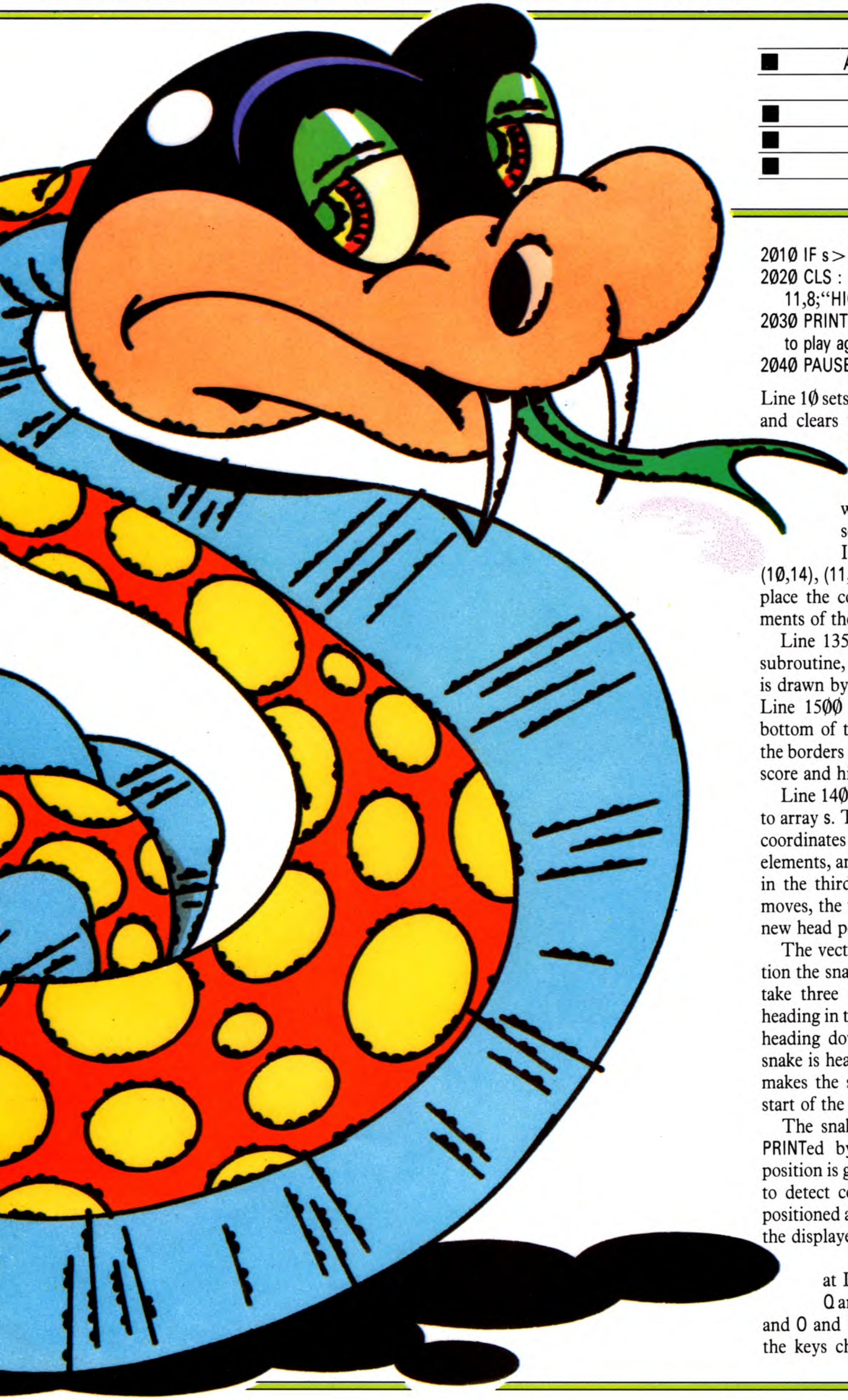
PLAYING THE GAME

The object of the game is to guide the hungry binary adder around the screen, gobbling up numbers which are plotted randomly. The numbers count down, so the longer you take with the snake, the lower your score will be. If you take too long, and the number decrements to zero, it will disappear, and another number will appear elsewhere. Eating a number will increase the snake's length by that number of segments.

Be careful not to overrun the border, nor allow the snake to cross over itself—particularly difficult as the snake gets longer. Crossing the border or the snake's body will end the turn, OUT will be displayed across the whole screen.

```
10 BORDER 1: PAPER 7: INK 9: CLS
20 LET hs=0
30 DIM s(570,2)
100 LET s=0
110 LET s(1,1)=10: LET s(1,2)=14
120 LET s(2,1)=11: LET s(2,2)=14
130 LET s(3,1)=12: LET s(3,2)=14
135 GOSUB 1500
140 LET t=1: LET h=3
145 LET yv=1: LET xv=0
150 FOR n=1 TO 3: PRINT PAPER 4;AT
   s(n,1),s(n,2);"#": NEXT n
160 LET y=12: LET x=14
165 LET p=0
170 GOSUB 1000
190 IF ATTR (y,x)<>56 AND ATTR
   (y,x)<128 THEN GOTO 2000
200 LET h=h+1: IF h=501 THEN LET h=1
210 PRINT PAPER 4;AT y,x;"#"
220 LET s(h,1)=y: LET s(h,2)=x
230 PRINT AT s(t,1),s(t,2);CHR$ 32
240 LET t=t+1: IF t=501 THEN LET t=1
250 IF p=0 THEN LET p=INT (RND*9)+1:
   LET fy=INT (RND*19)+2: LET fx=INT
   (RND*30)+1: IF ATTR (fy,fx)<>56
   THEN LET p=0: GOTO 250
260 PRINT PAPER INT (p/2); FLASH 1;AT
   fy,fx;p
270 IF RND<.98 THEN GOTO 290
280 LET p=p-1: IF p=0 THEN PRINT AT
   fy,fx;CHR$ 32
290 IF y<>fy OR x<>fx THEN GOTO 170
300 LET s=s+p: PRINT PAPER 4;AT
   y,x;"#": PRINT PAPER 6;AT 0,6;s
310 FOR n=1 TO p
320 GOSUB 1000
325 IF ATTR (y,x)<>56 THEN GOTO 2000
330 LET h=h+1: IF h=501 THEN LET h=1
340 LET s(h,1)=y: LET s(h,2)=x
350 PRINT PAPER 4;AT s(h,1),
   s(h,2);"#"
355 FOR m=1 TO 10: NEXT m
360 NEXT n
500 GOTO 165
1000 LET a$=INKEY$
1010 IF a$="q" THEN LET
   yv=-1: LET xv=0
1020 IF a$="a" THEN LET
   yv=1: LET xv=0
1030 IF a$="o" THEN LET xv=
   -1: LET yv=0
1040 IF a$="p" THEN LET
   xv=1: LET yv=0
1050 LET y=y+yv: LET x=
   x+xv: RETURN
1500 FOR n=22560 TO 22591:
   POKE n,16: POKE n+640,16:
   NEXT n
1510 FOR n=22592 TO 22592+
   32*19 STEP 32: POKE n,16:
   POKE n+31,16: NEXT n
1520 PRINT PAPER 1;AT 0,0;" SCORE□";
   PAPER 6;s; PAPER 1;TAB 14;"HIGH SCORE
   □"; PAPER 6;hs;PAPER 1;TAB 31;"□"
1590 RETURN
2000 PRINT AT 0,0;: FOR n=1 TO 88: PRINT
   FLASH 1; PAPER 2; INK 6;"OUT!"; PAPER
   6; INK 2;"OUT!";: BEEP
   .005,60-n:
   NEXT n
```





- A VERSION OF THE CLASSIC GAME WRITTEN IN BASIC
- PLOTTING FOOD
- EATING NUMBERS
- EXTENDING THE SNAKE

```
2010 IF s>hs THEN LET hs=s
2020 CLS : PRINT AT 8,10;"SCORE: ";s;AT
   11,8;"HIGH SCORE: ";hs
2030 PRINT INVERSE 1;AT 16,2;"Press any key
   to play again"
2040 PAUSE 0: CLS : GOTO 100
```

Line 1∅ sets the border, paper and ink colours and clears the screen. The high score and score are set to zero by Lines 2∅ and 1∅∅.

Line 3∅ DIMensions array s, which will be used to store the screen coordinates of the snake. Initially, the snake will occupy (1∅,14), (11,14) and (12,14). Lines 11∅ to 13∅ place the coordinates in the first three elements of the array.

Line 135 calls the 'setting up the screen' subroutine, starting at Line 15∅∅. The border is drawn by POKEing into the attribute file—Line 15∅∅ draws the lines at the top and bottom of the screen, and Line 151∅ draws the borders at the sides. Line 152∅ sets up the score and high score.

Line 14∅ sets up the head and tail pointers to array s. They tell the machine that the tail coordinates are stored in the first pair of array elements, and the head coordinates are stored in the third pair of elements. As the snake moves, the two pointers are adjusted and the new head position is slotted into the array.

The vectors xv yv, keep track of the direction the snake is heading. Both xv and yv can take three values: ∅ means the snake isn't heading in that direction; 1 means the snake is heading down or right; and −1 means the snake is heading up or right. Line 145, then, makes the snake move up the screen at the start of the game.

The snake, consisting of three hashes, is PRINTed by Line 15∅. The current head position is given by x and y and these are used to detect collisions. Line 16∅ has the head positioned at 14,12. Line 165 sets the value of the displayed number, p, to zero.

The INKEY$ subroutine, starting at Line 1∅∅∅, is called by Line 17∅. Q and A move the snake up and down, and O and P move it left and right. Pressing the keys change xv and yv, and Line 1∅5∅

changes the head position by adding xv and yv to x and v.

At the end of the subroutine, the program RETURNs to Line 190. Line 190 uses ATTR to check if the head is going to run into a square that isn't white (the background colour), or isn't flashing (a number). If both are true, the snake must have overrun the border, or crossed over itself, and the program jumps to Line 2000—the start of the 'end of game' routine.

Provided that the next head position is legal, Line 200 increments the head pointer. If the pointer now has a value outside the DIMensions of the array, the pointer is reset to one. The head is PRINTed at its new position by Line 210, and Line 220 sees that the new position is put into array s, at the elements indicated by the head pointer. The tail needs to be blanked out or else the snake will get longer and longer—Line 230 sees to this by referring to the array for the position at the tail. Line 240 increments the tail pointer, and checks, again if the pointer is pointing outside the DIMensions of the array.

If there isn't a number on screen, Line 250 chooses a number, p, and a random position for it. If the position isn't on a white area of screen—ATTR(fy,fx) < > 56—p is reset to zero, and another number and position is chosen. With a suitable position chosen, Line 260 PRINTs it flashing on the screen.

As time elapses, the number on screen is decremented. A slight random element is introduced in Line 270, by comparing a random number with .98. In most cases, the score decrementing line—Line 280—is jumped over. Line 280 checks that the number is still greater than zero after it's been decremented—otherwise the number is blanked out. If the head isn't occupying the same screen position as the number, the loop is completed by the program being sent back to Line 170.

The program reaches Line 300 if the snake has eaten the number. The score has the eaten number added to it. The head is PRINTed, and the score displayed. The FOR ... NEXT loop between Lines 310 and 360 adds the correct number of segments to the snake—the number of segments is determined by the value of the number the snake has eaten. Each time through the loop, Line 320 calls the INKEY$ routine; Line 325 checks if the head is still in a legal position; Line 330 increments the head pointer; Line 340 stores the head's new position in array s; and the head is PRINTed by Line 350. The extra segments are added by not blanking out the tail as it would be normally. Not blanking out the tail means that the snake would be seen to speed up if you didn't use the delay in Line 355.

The final section of program—starting at Line 2000—as mentioned earlier, is the 'end of game' routine. Line 2000 PRINTs 'OUT' all over the screen, at the same time making a series of BEEPs. Line 2010 updates the high score if it has been exceeded, and Line 2020 displays the score and high score. Lines 2030 and 2040 allow the player to start again.

```
10 POKE53280,2:POKE53281,1:PRINT
   "▢"
20 HS=0
30 DIM S(1000,2)
40 POKE54296,0
100 S=0
110 S(1,1)=10:S(1,2)=14
120 S(2,1)=11:S(2,2)=14
130 S(3,1)=12:S(3,2)=14
135 GOSUB1500
140 T=1:H=3
145 YV=1:XV=0
150 FORN=1TO3:SP=1024+S(N,1)*40+
   S(N,2):POKESP,163:POKE54272+SP,
   6:NEXT
160 Y=12:X=14
165 P=0
170 GOSUB 1000
190 SP=PEEK (1024+Y*40+X):IFSP=
   160ORSP=163ORY=0THEN2000
200 H=H+1:IFH=1000THENH=1
210 SP=1024+Y*40+X:POKESP,163:
   POKESP+54272,6
220 S(H,1)=Y:S(H,2)=X
230 SP=S(T,1)*40+S(T,2)+1024:POKE
   SP,32
240 T=T+1:IFT=1000 THEN T=1
245 IFP<>0 THEN260
250 P=INT(RND(1)*9)+1:FY=INT(RND
   (1)*21)+2:FX=INT(RND(1)*37)+1
255 IFPEEK(1024+FY*40+FX)<>32
   THEN250
260 SP=1024+FY*40+FX:POKESP,P
   +48:POKESP+54272,0
270 IF RND(1)<.95THEN 290
280 P=P-1:IFP=0THENPOKE1024+FX
   +FY*40,32
290 IFFY<>Y OR FX<>X THEN 170
300 S=S+P:SP=1024+Y*40+X:POKESP,
   163:POKESP+54272,6:PRINT"■■
   ■■■■■■"S
310 FORN=1TOP
320 GOSUB1000
325 SP=PEEK (1024+Y*40+X):IFSP=
   160ORSP=163ORY=0THEN2000
330 H=H+1:IFH=1000THENH=1
340 S(H,1)=Y:S(H,2)=X
350 SP=1024+S(H,1)*40+S(H,2):POKE
   SP,163:POKESP+54272,6
355 FORM=1TO10:NEXT
360 NEXT N
500 GOTO165
1000 GET A$
1010 IFA$="Q"THENYV=-1:XV=0
1020 IFA$="A"THENYV=1:XV=0
1030 IFA$="O"THENXV=-1:YV=0
1040 IFA$="P"THENXV=1:YV=0
1050 Y=Y+YV:X=X+XV:RETURN
1500 FORN=0TO39:POKE1024+N,160:
   POKE55296+N,0:POKE1984+N,160:
   POKE56256+N,0:NEXT
1510 FORN=1TO23:POKE1024+N*40,160:
   POKE1063+N*40,160
1515 POKE55296+N*40,0:POKE55335+
   N*40,0:NEXT
1520 PRINT"■■■ SCORE"S;TAB(18)
   "■HIGH SCORE ";HS
1590 RETURN
2000 FORF=0TO24:POKE54272+F,0:
   NEXT
2010 POKE54273,8
2020 POKE54277,175:POKE54278,30:
   POKE54296,15:POKE54276,129
2030 PRINT"■";TAB(18)"■■OUT!"
2040 FORF=0TO10:PRINT"■"TAB(18)
   "OUT!":FORG=0TO90:NEXT:PRINT
   "■"TAB(18)"■OUT!"
2050 FORG=0TO90:NEXTG,F:POKE54277,
   0:POKE54278,0
2060 IFS>HSTHENHS=S
2070 PRINT"▢■■■■■■
   ■■"TAB(15)"SCORE"S:PRINT
   "■■"TAB(13)"HIGH SCORE"HS
2080 PRINTTAB(6)"■■■PRESS ANY
   KEY TO PLAY AGAIN."
2090 POKE198,0:WAIT198,1:PRINT
   "▢":GOTO100
```

The main program listing given above is for the Commodore 64, but if you have a Vic with 3K RAM Expansion you can still play the game by changing the lines shown below, and not using Lines 2010 and 2020. The size of the playing area has had to be limited because of the limited memory size.

```
10 POKE36879,26:PRINT"▢"
30 DIM S(308,2)
40 POKE36878,15
135 POKE 36867,32:GOSUB1500
140 T=1:H=3
145 YV=1:XV=0
150 FORN=1TO3:SP=7680+S(N,1)*22+S
   (N,2):POKESP,163:POKE30720+SP,
   6:NEXT
190 SP=PEEK (7680+Y*22+X):IFSP=160
   ORSP=163ORY=0THEN2000
200 H=H+1:IFH=308THENH=1
210 SP=7680+Y*22+X:POKESP,163:POKE
   SP+30720,6
```

```
230 SP=S(T,1)*22+S(T,2)+768Ø:POKE
    SP,32
240 T=T+1:IFT=3Ø8 THEN T=1
245 IFP<>Ø THEN26Ø
25Ø P=INT(RND(1)*9)+1:FY=INT(RND(1)
    *13)+2:FX=INT(RND(1)*18)+1
255 IFPEEK(768Ø+FY*22+FX)<>32THEN
    25Ø
26Ø SP=768Ø+FY*22+FX:POKESP,P+48:
    POKESP+3Ø72Ø,Ø
28Ø P=P-1:IFP=ØTHENPOKE768Ø+
    FX+FY*22,32
3ØØ S=S+P:SP=768Ø+Y*22+X:POKESP,
    163:POKESP+3Ø72Ø,6:PRINT"■■"S
325 SP=PEEK (768Ø+Y*22+X):IFSP=16Ø
    ORSP=163ORY=ØTHEN2ØØØ
33Ø H=H+1:IFH=3Ø8THENH=1
35Ø SP=768Ø+S(H,1)*22+S(H,2):POKE
    SP,163:POKESP+3Ø72Ø,6
15ØØ FORN=ØTO21:POKE768Ø+N,16Ø:
    POKE384ØØ+N,Ø:POKE8Ø1Ø+N,16Ø:
    POKE3873Ø+N,Ø
151Ø POKE 768Ø+N*22,16Ø:POKE77Ø1
    +N*22,16Ø
1515 POKE384ØØ+N*22,Ø:POKE38421
    +N*22,Ø:NEXT
152Ø PRINT"■■■"S;TAB(1Ø)HS
2ØØØ FOR Z=2ØØTO127STEP-1:POKE
    36877,Z:NEXT
2Ø3Ø PRINT"■■■";TAB(9)
    "■■OUT!"
2Ø4Ø FORF=ØTO1Ø:PRINT"■■■"
    TAB(9)"OUT!":FORG=ØTO9Ø:NEXT:
    PRINT"■■■"TAB(9)"■OUT!"
2Ø5Ø FORG=ØTO9Ø:NEXTG,F
2Ø7Ø POKE 36867,174:PRINT"■■■■
    ■■■■■■■■■■
    SCORE"S:PRINT"■■HIGH SCORE"HS
2Ø8Ø PRINT"■■PRESS ANY KEY TO
    PLAY□"
2Ø9Ø POKE198,Ø:WAIT198,1:PRINT
    "■":GOTO1ØØ
```

Line 1Ø sets up the screen colours and clears the screen ready for the game's graphics. The high score is set to zero in Line 2Ø, and the score is set to zero by Line 1ØØ.

Line 3Ø DIMensions array S, which will be used to store the screen coordinates of the snake. The maximum length of the snake is 1000 segments for the Commodore 64 and 308 for the Vic, so a two-dimensional array, 1000 by 2, or 308 by 2 elements is needed because x and y coordinates are needed to define each screen location. Initially, the snake will occupy (1Ø,14), (11,14) and (12,14). Lines 11Ø to 13Ø place the coordinates in the first three pairs of elements in the array.

The 'setting up the screen' subroutine, starting at Line 15ØØ, is called by Line 135. Line 15ØØ draws the borders at the top and bottom of the screen, and Lines 151Ø and 1515 draw the borders at the sides. The score and high score are set up by Line 152Ø.

H and T in Line 14Ø are pointers to the array. H points to the pair of elements which hold the head's coordinates, and T does the same for the tail's coordinates. In the case of Line 14Ø, the tail is stored in the first pair, and the head is stored in the third pair.

Line 145 sets two vectors—xv and yv—which keep track of which direction the snake is moving. Both can take one of three values—Ø means the snake isn't heading in that direction; 1 means that the snake is heading down or right; and −1 means the snake is heading up or right. Line 145, then, makes the snake move up at the start of the game.

Line 15Ø POKEs the snake on to the screen. There's a FOR ... NEXT loop from one to three, which makes the snake start off consisting of three segments. Line 16Ø uses two variables—X and Y—to keep track of the snake's head. The two variables are set to the position that the head has been POKEd into in Line 15Ø. Line 165 contains a flag, P, which indicates if a number is being displayed.

Line 17Ø calls the keyboard reading subroutine starting at Line 1ØØØ. Lines 1Ø1Ø to 1Ø4Ø read the Q, A, O and P keys, and set the vectors appropriately. Line 1Ø5Ø uses the vectors to calculate the position of the snake, and ends the subroutine.

Line 19Ø checks if the snake has overrun the borders, or has crossed over itself. If it has, it jumps to the 'end of game' routine at Line 2ØØØ. If the snake's position is legal, Line 2ØØ increments the head pointer, and moves it back to the beginning of the array if the end has been reached. The head is POKEd into its new position by Line 21Ø, and Line 22Ø enters the new position into the array. Line 23Ø blanks out the tail, so that the snake doesn't get longer and longer. Line 24Ø increments the tail pointer, and checks that the pointer is still pointing to an element within the array.

If a number is being displayed, Line 245 sends the program to Line 26Ø. If not, a new number will have to be displayed. Line 25Ø chooses a number at random, and a position for it. Line 255 checks if the position chosen isn't clear space. If it isn't, the program goes back to Line 25Ø again. Line 26Ø POKEs the number on to the screen.

As time elapses, the number on the screen is decremented. A slight random element is introduced in Line 27Ø, by comparing a random number with Ø.95. In most cases, the score decrementing line—Line 28Ø—is jumped over. Line 28Ø checks that the number is still greater than zero after it's been decremented—otherwise the number is blanked out. If the head isn't occupying the same screen position as the number, Line 29Ø completes the loop, by sending the program back to Line 17Ø.

If the snake has eaten the number, the program reaches Line 3ØØ. The score is increased by the number that's just been eaten. The head is POKEd on screen, and the score displayed. The FOR ... NEXT loop between Line 31Ø and 36Ø adds the correct number of segments to the snake—the number of segments is determined by the number that



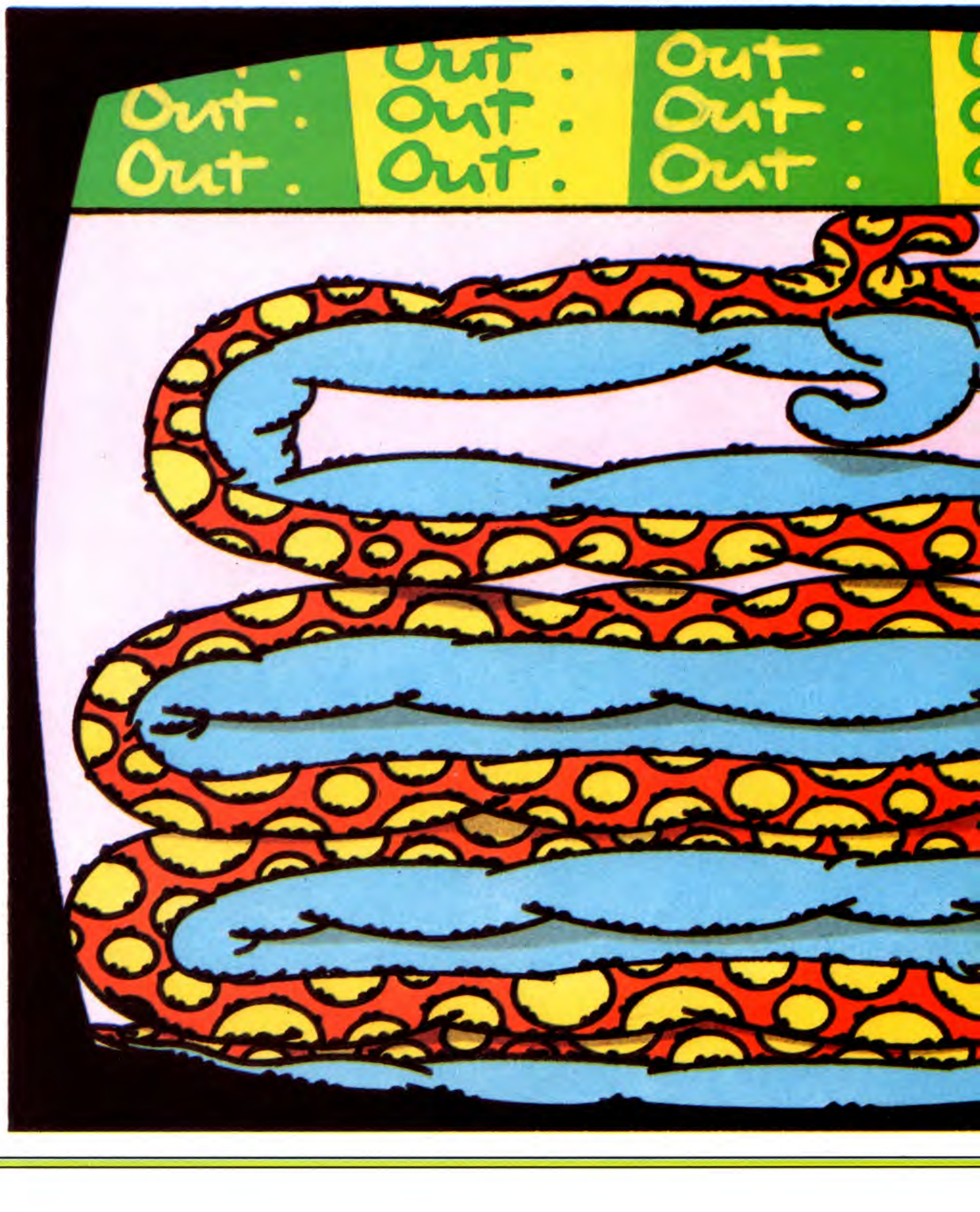
the snake has got its fangs round. Each time through the loop, Line 32Ø calls the keyboard reading routine; Line 325 checks if the head is still in a legal position; Line 33Ø increments the head pointer; Line 34Ø stores the head's new position in the array; and the head is POKEd by Line 35Ø. The extra segments are added by not blanking out the tail. With less to do, the program would speed up the progress of the snake, but introducing the delay in Line 355 keeps the speed of the snake constant.

The final section of the program, starting at Line 2ØØØ is the 'end of game' routine. In the case of the Commodore 64, Line 2ØØØ makes the machine ready to generate a sound effect. Lines 2Ø1Ø and 2Ø2Ø generate the sound effect. In the case of the Vic 20, the sound effect is generated in Line 2ØØØ alone. Don't forget to turn the sound on your TV up. Lines 2Ø3Ø and 2Ø4Ø flashes the word OUT! on the top border. There's a short pause in Line 2Ø5Ø before the sound is turned off. Line 2Ø6Ø updates the high score if necessary, and Line 2Ø7Ø displays the score and high score.

NOTE: If you have a BBC fitted with a disk drive unit, change to MODE 4 in Line 2Ø.

```
10 *FX11
20 S%=0:MODE1
30 VDU23;8202;0;0;0;
40 DIM P(39*31)
50 PROCSCREEN
60 P(660)=-40:P(620)=-40
70 F=0:N=0:V=0:V2=0:SC=0:HISC=0
80 X%=20:Y%=15
90 D=-40:H=580:T=660
100 PRINTTAB(20,15)"*"TAB(0,1)"SCORE"
110 PROCCOM:IF F=1 THEN 260 ELSE 110
120 DEF PROCSCREEN
130 CLS:COLOUR131
140 FOR Q=0 TO 39:PRINTTAB(Q,2)
   "□"TAB(Q,29)"□":P(2*40+Q)
   =1:P(29*40+Q)=1:NEXT
150 FOR Q=3 TO 28:PRINTTAB(0,Q)
   "□"TAB(39,Q)"□":P(Q*40)=1:
   P(Q*40+39)=1:NEXT
160 COLOUR128:ENDPROC
170 DEF PROCCOM
180 TD=D:K=INKEY(1):D=40*(K=80)
   -40*(K=76)+(K=90)-(K=88):IF
   D=0 THEN D=TD
190 PRINTTAB(X%,Y%)"#":X%=
   (H+D) MOD 40:Y%=(H+D) DIV 40:
   PRINTTAB(X%,Y%)"*"
200 P(H)=D:H=H+D
210 IF P(H)>100 THEN V=P(H)-
   100:N=0:GOTO 230
220 IF ABS(P(H))=40 OR ABS(P
   (H))=1 THEN F=1:ENDPROC
230 PRINTTAB(T MOD 40,T DIV 40)"□"
240 IF V>0 THEN V=V-1:SC=SC+1
   ELSE TT=T:T=T+P(T):P(TT)=0
250 PRINTTAB(10,1);SC:PROCJ:ENDPROC
260 CLS:FOR T=1 TO 500:PRINT
   "OUT□";:NEXT:CLS
270 PRINTTAB(10,10)"SCORE□□□
   □□□=";SC
280 IF S%<SC THEN S%=SC :PRINT
   TAB(7,5)"YES IT'S A HIGH SCORE"
290 PRINTTAB(10,13)"HIGH
   SCORE□=";S%
300 PRINTTAB(0,28)"PRESS ANY KEY FOR
   ANOTHER ATTEMPT"
310 *FX15,0
320 G=GET:CLEAR:GOTO 20
330 DEF PROCJ
340 IF N=0 THEN 380
350 IF V=0 AND RND(1)<.02 THEN
   V2=V2-1:P(X+40*Y)=V2+100
360 IF V2=0 THEN P(X+Y*40)=0:
   N=0:PRINTTAB(X,Y)"□" ELSE
   COLOUR131:COLOUR0:PRINTTAB
   (X,Y);V2:COLOUR3:COLOUR128
370 ENDPROC
380 IF V<>0 THEN ENDPROC ELSE
   V2=RND(9):X=RND(38):Y=RND(24)+3
390 IF P(X+40*Y)=0 AND X+40*Y
   <>H THEN COLOUR131:COLOUR0:
   PRINTTAB(X,Y);V2:COLOUR3:COLOUR
   128:P(X+Y*40)=V2+100 ELSE 380
400 N=1:ENDPROC
```

At the start of the program, Line 2Ø sets S%, the high score variable, to zero. The game takes place on the MODE 1 screen—set up by Line 2Ø—so there are 40 screen positions across the width, and 32 top to bottom. Line 3Ø turns off the cursor, and Line 4Ø DIMensions the array P, which will be used to store the screen coordinates of the snake's



head and tail. The array has to be DIMensioned so that it can contain the maximum length of the snake, but because the array starts from element zero, the DIMensions are 39*31.

Line 5Ø calls PROCSCREEN, which starts at Line 12Ø. Line 13Ø clears the screen and defines its COLOUR—COLOUR 131 means the spaces PRINTed later will appear in white. Line 16Ø changes the COLOUR black.

Line 6Ø sets the values of two elements in P. Setting array elements to −4Ø means that the next segment of the snake—going towards the head—is one line above the segment. If the value is 4Ø, the next segment is below, and similarly if the value is −1, the segment is to the left, and if it is 1, the next segment is to the right. The values stored in P are simply pointers to the next segment. In the case of Line 6Ø, the snake is positioned centrally, and is moving up the screen initially.





Line 7Ø sets a pair of flags, a pair of variables and the score to zero. F is the dying flag, and is set if the snake hits the wall or crosses over its body, N is set to one if there's a number on screen, and V and V2 are used to manipulate the number.

In Line 8Ø, X% and Y% are the coordinates of the snake's head. The direction the head is moving is the value of variable D, and the array elements corresponding to the snake's head and tail are the values of variables H and T. Line 1ØØ PRINTs the head and SCORE.

Line 11Ø calls PROCCOM, which starts at Line 17Ø. Line 18Ø reads the keyboard. Z and X move the snake left and right, and P and L move it up and down. The value of D is derived from the keypresses by using Boolean logic. TD is simply a temporary store for the direction—Temporary Direction.

Line 19Ø PRINTs a body segment over the head's last position as part of the animation of the snake. The remainder of the line works out where the head is now, and then PRINTs it at its new position. Line 2ØØ puts the head direction in the array at P(H), before adding D to the head pointer, so the program knows the head's new position.

Line 21Ø looks at the array element corresponding to the new head position. If the number stored in that element is greater than 1ØØ, the corresponding screen position must contain a number—in other words, the snake must have eaten a number, and the score will need to be increased, and the snake's length increased also. The program handles these flashing numbers by storing that number, plus 1ØØ, in the array element corresponding to the screen position. To find out the value of the number that's been eaten, the program needs to subtract 1ØØ from the stored number—the value of the number is V. The number flag, N, is reset to zero, and the program jumps to Line 23Ø.

If the snake hasn't eaten a number, Line 22Ø looks to see if it has crossed over itself—it looks to see if the value in the array element is 1, −1, 4Ø or −4Ø. Using ABS—absolute—saves having to examine all four values. If it has crossed over itself, the dying flag, F, is set to one, and the PROCedure ends.

If the snake has successfully eaten a number, the program reaches Line 23Ø. If the number's value is still greater than one, the tail is blanked out, the value of the number is decremented, and the score incremented. If the number's value has decreased to zero, the tail is moved on in the array. The score is PRINTed, and PROCJ is called in Line 25Ø.

PROCJ starts at Line 33Ø and ends at Line 4ØØ. If no number is being displayed—N=Ø—Line 34Ø sends the program to Line 38Ø. If there is a number on screen, Line 35Ø decides whether to decrement the number, and does so, if it needs to. If the number is zero, Line 36Ø, enters zero in the correct place in the array and blanks out the number on screen. The number flag is reset to zero. The PROCedure ends at Line 37Ø.

The Lines from 38Ø to 4ØØ deal with plotting a new number on screen, if there is no longer one being displayed—if N=Ø. The start of Line 38Ø is an extra check on the value of V. Next, a random value for V2 is chosen, and random X and Y coordinates for plotting it on screen. Line 39Ø checks if zero is stored in the array at the coordinates chosen, and that the coordinates do not correspond to the position of the head. Provided that both the conditions are satisfied, the number is displayed on screen. If the conditions aren't satisfied, the program chooses another random number, and a new position for it. The number flag is set to one in Line 4ØØ, and the PROCedure ends.

All that remains of the program now, is the routine from Line 26Ø to Line 32Ø. The program reaches Line 26Ø if the snake overruns the border, or crosses its own body, and PRINTs OUT across the screen, then clears it. The score is displayed by Line 27Ø, and the high score—S%—updated if necessary by Line 28Ø. Finally, Lines 3ØØ to 32Ø are an 'another go?' routine.

```
1Ø M=512:DIM B(M)
2Ø GOSUB 6ØØ
3Ø GOTO 5ØØ
1ØØ K$=INKEY$:IFK$=""THEN
   K$=L$
11Ø K=ASC(K$):NP=B(H)−32*(K=1Ø)
   +32*(K=94)−(K=9)+(K=8)
12Ø TE=PEEK(NP)
13Ø IF TE=FG OR TE=HC OR TE=BC OR
   NP<1Ø56 THEN POKE NP,HC:E=1
14Ø L$=K$:RETURN
```

```
200 R=RND(9):RX=RND(13)+1:RY=
    RND(29)+1:RP=RX*32+RY
210 IF PEEK(1024+RP)<>BG THEN 200
220 R$=RIGHT$(STR$(R),1):PRINT@
    RP,R$;:P=1:RETURN
250 R=R-1:IF R=0 THEN PRINT@RP,
    CHR$(BG);:P=0:RETURN ELSE PRINT
    @RP,RIGHT$(STR$(R),1);:RETURN
300 SC=SC+R:PRINT@26,SC;:P=0
310 SL=SL+1:POKE NP,BC
320 H=H-1:IF H=0 THEN H=M
330 B(H)=NP
340 FOR J=1 TO R
350 PLAY "L255CEF"
360 GOSUB100:IF E=1 THEN 800
370 H=H-1:IF H=0 THEN H=M
380 B(H)=NP:POKE NP,BC
390 NEXT
400 POKE NP,HC:RETURN
500 IF P=0 THEN GOSUB200 ELSE IF
    RND(150)<SL THEN GOSUB250
510 GOSUB100:IF E=1 THEN800
520 IF TE=R+112 THEN POKE B(H),
    BC:GOSUB300
530 POKE B(H),BC:POKE B(T),BG
540 H=H-1:IF H=0 THEN H=M
550 T=T-1:IF T=0 THEN T=M
560 B(H)=NP:POKE NP,HC
570 GOTO500
600 BG=RND(7)+1:FG=RND(7)+1:
    L$=CHR$(94):P=0:SC=0:SL=1:
    E=0
610 IF BG=FG THEN600
620 CLS BG:BG=BG-1:FG=FG-1
630 BG=143+16*BG
640 FG=143+16*FG
650 FOR J=1024 TO 1055:PLAY
    "T255L255O4AG":POKE J,FG:
    POKEJ+480,FG:NEXT:FORJ=1056
    TO 1472 STEP 32:PLAY"O2DA":
    POKEJ,FG:POKE J+31,FG:NEXT
660 PRINT@3,"HIGH=";HS;:PRINT
    @15,"YOUR SCORE=□0□";
670 HC=ASC("*")+64:BC=ASC
    ("#")+64:T=3:H=1
680 B(1)=1263:POKE B(1),HC
690 B(2)=1295:POKE B(2),BC
700 B(3)=1327:POKE B(3),BC
710 RETURN
800 CLS RND(8):PLAY"O1ACDEFG
    ACDEFG":FORK=1TO408:PRINT
    "OUT□";:NEXT:PLAY"O4ABCDEFG
    O3ABCDEFGO2ABCDEFG"
810 IF HS<SC THEN HS=SC
820 CLS:PRINT@73,"SCORE□□□=";
    SC:PRINT@230,"HIGH SCORE□=";
    HS
830 A$=INKEY$:PRINT@450,"PRESS A KEY
    TO PLAY AGAIN"
840 A$=INKEY$:IF A$="" THEN 840 ELSE
    20
```

The Snake game takes place on the text screen, rather than the high resolution screen which you have been using for a large number of games in *INPUT*. The game doesn't suffer, though—the text screen has been chosen simply because it's the most logical way to write the game, with characters representing segments of the snake.

In Line 10 the array **B** is **DIM**ensioned so that it can accommodate the maximum size of the snake—512 character squares. Note that each element of the array doesn't correspond to a character square on the screen, but is just a box in which the coordinates of part of the snake can be stored. Line 20 jumps to Line 600 which is the initialization subroutine.

In Line 600 **BG** is the background colour and **FG** is the foreground (border) colour. **L$** is part of the way that the program handles keyboard input. Setting **L$** to **CHR$(94)** means that the snake always starts off moving up the screen unless a key has been pressed previously—but more about controlling the snake a little later. **P=0** tells the machine that no number is being displayed, **SC=0** means that the score is zero, **SL=1** means that the skill level is one, and **E=0** tells the machine that the snake is still within the playing area, and isn't trying to cross over its own body. The three flags are used during the program so that various subroutines can be called at the correct times.

Line 610 makes sure the foreground and background colours are different—two new colours are chosen if the original pair are identical. Line 620 uses **CLS BG** to colour the screen in the background colour. **BG** and **FG** have one subtracted from them ready for the calculations in Line 630 and 640. The calculations may look quite complex, but all they do is to convert the values of **BG** and **FG** into ones that can be **POKE**d on the screen, drawing the borders and blanking out the tail of the snake as it moves across the screen. Line 650 draws the border, and **PLAY**s a few notes as it does so. The score and high score panels are also drawn.

In Line 670, **HC** is the head code, and **BC** the body code. **H** and **T** are pointers to the array, **B**, representing the head and tail of the snake. Lines 680 to 700 **POKE** a head and two body segments on to the screen, so that the snake consists of a head and two body segments at the start of each turn.

The program **RETURN**s to Line 30, which jumps to the main loop of the program—from Line 500 to 570. Line 500 first checks if a number is being displayed by looking at the value of **P**, the number display flag. If **P** is zero, the subroutine starting at Line 200 is called. If there is a number on screen, a random number between 1 and 150 is generated and compared with the current skill level. If the number is lower than the skill level, the program jumps to the subroutine starting at Line 250.

The two alternative subroutines work like this. The subroutine starting at Line 200 plants a random number at a random position on screen. Line 200 chooses the number, **R**, between 1 and 9, and chooses an X and Y coordinate for positioning the number. **RX** and **RY** are used for calculating **RP**, the screen position. Before the number is plotted on screen, Line 210 checks that the position contains the background colour—i.e. it's not on the border, or the snake itself. Line 220 **PRINT**s the number on screen and sets the number displayed flag. The second subroutine, consisting of just one line—Line 250—decrements the number on screen. If the number becomes zero when decremented, the screen position occupied by the number is **PRINT**ed over in the background colour—the number is **PRINT**ed out, in other words. If the number doesn't become zero, the new value is displayed in the same screen location as the earlier value. The subroutine **RETURN**s to Line 510.

Line 510 calls yet another subroutine—this time it's the one starting at Line 100 that has been used earlier in the program to read the keyboard. The **IF . . . THEN** checks if the snake has run out of screen, or crossed over itself. If it has, the program jumps to Line 800—the routine leading to the end of the game.

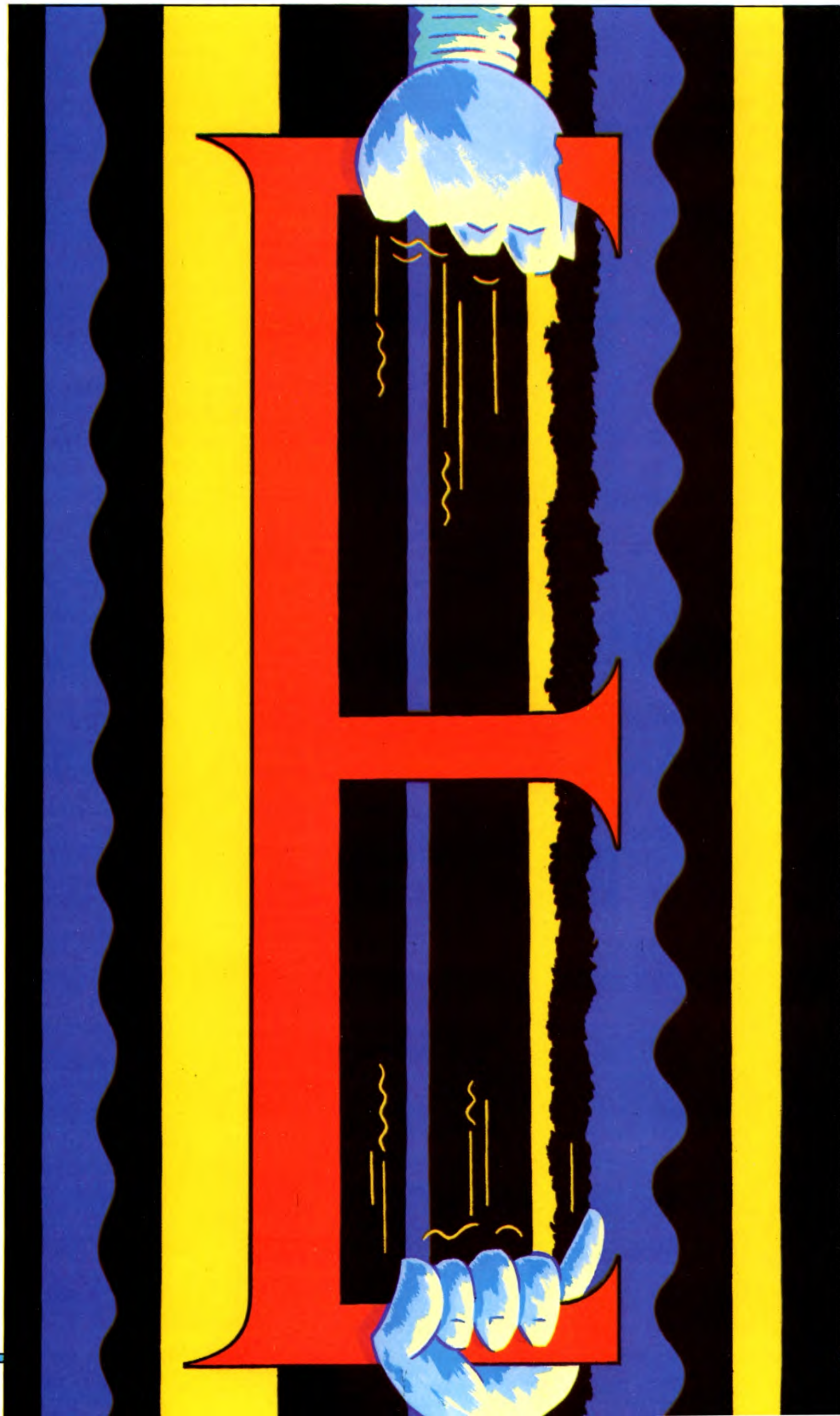
Line 520 tests if a number has been successfully swallowed by adding 112 to **R**, and comparing the result with **TE**. You have to add 112 here because **TE** is derived from **PEEK**ing the screen, and the value returned after **PEEK**ing isn't the same as the number that is being displayed. If a number has been swallowed, it's blanked out using the body code, then the subroutine at Line 300 is called. The score is increased, and another number is plotted on screen.

Lines 540 and 550 change the head and tail pointers, and adjust them if they have dropped to zero. Line 560 **POKE**s the snake's head on to the screen.

Sitting at the end of the program, starting at Line 800, is the 'game ends' routine. Line 800 clears the screen to a randomly chosen colour. A short tune is **PLAY**ed before the **FOR . . . NEXT** loop fills the screen with the word OUT. There's another short tune after the X loop. The high score is updated if necessary by Line 810, before the 'another go?' routine in Lines 820 to 840. Opting for another go will send the program back to Line 20.

MAKING THE HEADLINES

- ENLARGING ROM CHARACTERS
- DOUBLE HEIGHT AND DOUBLE WIDTH
- DESIGNING YOU OWN BLOCK LETTERS



If you want to set up a striking title page or other prominent display you'll need to create a special typeface. Here are two methods you can try

The computer's normal on-board character set leaves a lot to be desired. After all, it was designed for economy of space. So if you want something that looks more like the headline than the small print, you need to set up a special typeface.

There are several ways in which you can call on the computer's standard graphics facilities to enable you to create custom display letters. And each of these can be used for different types of effect. For example, you could DRAW the letters out line by line, or perhaps build them up from block graphics.

Of course, you can use any of these methods within a program—so that the first 2Ø lines might set up 'INPUT PRESENTS', for example—but this is generally an uneconomic way of doing things, not to mention difficult to work out.

A far better way, particularly if you are going to want display letters quite frequently, is to set up a separate program to generate them. Once this has been done, you can simply enter the text you want to use, and let the computer plot the letters automatically. Then all you have to do is to SAVE the display out of the letter generator and into the program in which you want to use them.

This article is the first of two which explain the various ways to do this, and give listings for the programs you need. In this part you will see two elegant solutions to the problem. The first program simply scans the character set in the computer's memory, and enlarges it to a display size. You get a typeface which is similar to the normal characters, but much more prominent. And you pick your letters simply by typing them in, in the normal way. Unfortunately, this method is not possible on the Dragon and Tandy as you cannot get access to the ROM character set. The second program builds up giant size characters from block graphics and works on all the computers.

In the next article you will see how to create yet another style of lettering which can be adapted for all sorts of applications. You will also find out how you can use your display to brighten up another program.

USING THE PROGRAMS

When you RUN the program, the computer waits for you to INPUT a string. The Spectrum and Commodores wait for you to enter a piece of text, and then they PRINT it out on the screen in double height letters.

The Acorn program first expects you to INPUT either H, for double height letters, or W, for double width letters, or both, for double height and double width letters. It then lets you enter a length of text, which it PRINTs on the screen in the form you choose.

For all of the computers, it is probably a good idea to enter a short word to start with, so that you can see the effect. The reason for this is that if you enter too long a string, the words will be split over the ends of the line.

Each of the programs actually looks at the bytes stored in the computer's ROM, and then uses these to define UDGs in various forms to make up each large letter. For example, with the double height letters, the computer replaces every byte from the ROM character set with two bytes in one of the UDGs.

```
1ØØØ INPUT "ENTER ANY LENGTH TEXT",
   LINE I$: CLS : GOSUB 9ØØØ: GOTO 1ØØØ
9ØØØ LET line=Ø: LET col=-1
9Ø1Ø LET y=Ø
9Ø2Ø FOR i=1 TO LEN I$: LET col=col+1:
   IF col=32 THEN LET col=Ø: LET
   line=line+2
9Ø3Ø LET t$=I$(i)
9Ø5Ø FOR x=Ø TO 6 STEP 2
9Ø6Ø POKE USR "a"+x,PEEK (15616+
   (8*(CODE t$-32))+y)
9Ø7Ø POKE USR "a"+1+x,PEEK
   (15616+(8*(CODE t$-32))+y)
9Ø8Ø LET y=y+1
9Ø9Ø NEXT x
91ØØ FOR x=1 TO 7 STEP 2
911Ø POKE USR "b"+x-1,PEEK
   (15616+(8*(CODE t$-32))+y)
912Ø POKE USR "b"+x,PEEK
   (15616+(8*(CODE t$-32))+y)
913Ø LET y=y+1
914Ø NEXT x
915Ø LET y=Ø
916Ø PRINT AT line,col;CHR$ 144;AT
   line+1,col;CHR$ 145
918Ø NEXT i
92ØØ RETURN
```

The first line in the program lets you enter your text, and puts it into the string variables I$. Then the computer sets up two variables for the position on the screen at which the computer will PRINT your text. They start off at Ø, for the vertical coordinate, and −1 for the horizontal one (in a moment, you will see why it is set to −1). Then a third variable, y, is set equal to Ø, before the computer actually starts to enlarge your text.

Line 9Ø2Ø starts a FOR ... NEXT loop equal to the number of characters you have entered, and then increases the column position (the variable col) by 1. This is why it was set to −1 to start with, since to make the first letter start flush with the left hand side of the screen, (at column Ø), the variable has to become Ø.

If the column variable is 32 (one more than the last position on each line) it is reset to Ø, so that the computer PRINTs any more letters at the start of a new line. The LINE variable is also increased, by two, to move the position down so that new letters are PRINTed underneath the old ones.

Line 9Ø3Ø sets up a string variable, t$, equal to the letter which the computer is about to stretch. It does this by using the control variable of the FOR ... NEXT loop (i) and setting t$ as equal to the i'th character in the string you entered.

Then the computer starts another FOR ... NEXT loop, which POKEs two bytes into a UDG, four times.

The calculation involved in finding out what the computer POKEs into the UDG is the key to expanding the letters.

ENLARGING THE LETTERS

The first half is simple. It POKEs an address x places after the first byte of UDG a. The variable x is the control variable of the new FOR ... NEXT loop, and increases in STEPs of two, so that the second byte which the loop POKEs in is not changed when the computer RUNs through the loop again.

The second half, which works out what is to be POKEd in, PEEKs the relevant byte in the current character from the ROM character set. 15616 is where the computer's own ROM characters start in memory. The calculation then takes away 32 from the CODE number of the character being stretched and multiplies it by 8, to find how far into the character set the letter being expanded is.

It is necessary to take 32 away from the CODE number before multiplying it by 8 because of the way that the Spectrum stores its characters (in fact, the first 32 character codes do not have 8 bytes reserved for them in this part of memory).



When the computer has worked out how far the character is into the character set, it adds this number to the first address of the ROM character set, 15616, and then adds y to it. y is the variable which is set to Ø at the beginning of the program. And it will be used to tell the computer which byte within the current character is being PEEKed, and POKEd. Adding Ø as here, just means that it PEEKs the first byte of the current character.

In Lines 9Ø6Ø and 9Ø7Ø, the computer POKEs *two* addresses in the UDG area with the contents of just *one* address in the ROM character set: thus doubling the height of each letter. After it has done the first two POKEs, it increases y by one so it moves on to the next byte, and then repeats this routine until the FOR ... NEXT loop has finished—at which point all eight bytes have been POKEd.

Then the Spectrum starts another FOR ... NEXT loop which does the same, but this time for the second UDG ("b") instead of UDG "a". This loop ends in Line 914Ø, and the computer resets y to Ø.

Line 916Ø actually PRINTs the UDGs, one above the other. It uses the variables line and col to tell it the position at which it should PRINT the characters. The program does not PRINT "A", but uses CHR$ 144 instead, as otherwise, you would not be able to distinguish between a normal A and a graphics A.

Finally, in Line 918Ø, the computer finishes the main FOR ... NEXT loop, and goes back to expand the next letter in your string. If it has already expanded, and PRINTed, all the letters you typed in, the computer goes back to Line 1ØØØ to let you enter another string.

```
1 PRINT"PLEASE WAIT—LOADING
   GRAPHICS. . ."
1ØØ POKE56334,PEEK(56334)AND254
11Ø POKE1,PEEK(1)AND251
116 K=-1
117 FORT=ØTO511:POKE12288+T,
   PEEK(53248+T):NEXT
118 FORI=512TO2Ø47:POKE12288+1,Ø:
   NEXT
13Ø FORC=ØTO52STEP2
135 K=K+1
14Ø FORT=16TOØSTEP-2
16Ø POKE12288+((C+64)*8)+T,
   PEEK(53248+K*8+(T/2))
17Ø POKE12288+((C+64)*8)+T-1,
   PEEK(53248+K*8+(T/2))
18Ø NEXT:NEXT
185 FORT=1TO26
186 POKE12831+X,Ø:X=X+16:NEXT
19Ø POKE1,PEEK(1)OR4
2ØØ POKE56334,PEEK(56334)OR1
25Ø PRINT"▣":FORT=55456TO55536:
   POKET,1:NEXT
3ØØ POKE53272,29
31Ø PRINT"▤▣▣▣▣▣▣▣▣▣
   INPUT A NAME MAX.4Ø CHARS."
32Ø C=Ø:X=Ø:INPUTA$
33Ø IFLEN(A$)>4ØTHEN25Ø
34Ø FORT=1TO2
35Ø FORY=1TOLEN(A$):B$=MID$
   (A$,Y,1):B=ASC(B$)
355 B=B-64
36Ø IFB$="□"THENPOKE1184+X,
   32:X=X+1:NEXT
37Ø POKE1184+X,64+(B*2-C):X=X+1
38Ø NEXT
39Ø X=4Ø:C=-1:NEXT
4ØØ GETA$:IFA$=""THEN4ØØ
41Ø GOTO25Ø
```

The Commodore and Vic programs work slightly differently from those for the Acorn and Spectrum, since the Commodore computers do not have any 'ready to use' UDGs. So, the Commodore programs begin by copying the ROM character set into RAM, where the computer can alter them. This is done by Lines 1ØØ to 2ØØ. If you are not quite sure as to what each POKE does, you should read the article on pages 450 to 457.



The character set in RAM is changed so that some of the characters can be used for double height characters. The Lines 130 to 180 use two FOR ... NEXT loops to double each letter in the alphabet, and store the two characters for each double height letter in the character set in RAM.

The computer actually doubles the height by PEEKing the bytes of the normal character, and POKEing each into a new character twice.

When the computer gets to Line 300 it changes the character set pointer to point to the new set in RAM, and then lets you INPUT a string, which it will PRINT in double height characters. It also sets two variables, X and C, equal to 0. These two variables are used in the next few lines to make the computer look at the right place in memory for the double height characters.

Line 330 checks that your string is 40 characters or less. If it is over this length it goes back to Line 310.

READING THE LETTERS

The loop FOR T=1 TO 2, in Line 340, is so that the computer runs through every letter in your string twice, once for each half of the letter. The loop in the next Line makes the computer execute the next few lines once for every letter in your string.

Then, B$ is set equal to the next letter in your string, using the function MID$. B is set equal to the ASCII code of the character in B$, and then the computer subtracts 64 from B so that it can be used in Line 370 to determine which letter is PRINTed. Line 360 checks to see whether the current character in B$ is a space. If it is, the computer POKEs a space onto the screen, and does a NEXT to start on the next letter.

So the computer only reaches Line 370 if B$ does not contain a space. This line also POKEs a character onto the screen—if the computer is PRINTing the top half, then Line 370 directs it to the top half of whichever character is in B$, and if it is working on the bottom half, Line 370 points to the bottom half of the character in B$.

It is in this Line that the variables X and C are used. X is added onto the location on the screen to be POKEd—so it is really changing the PRINT position. C is used to switch between the different character sets (for the upper and lower halves of the double height alphabet).

After POKEing the character onto the screen in Line 370, the computer does a NEXT to go on to the next letter. When it has finished all the letters in your string, it goes to Line 390, where X is set to 40 (to bring the PRINT position to the start of the next Line) and C is set to −1. The NEXT in Line 390 refers to the FOR ... NEXT loop with T as its control variable. All this did was make the computer PRINT two lines of characters, for the two halves of the double height characters.



Double height letters are ideal for a title page of a game or program

Line 400 waits for you to press a key before letting the computer go to Line 410, which sends the computer back to Line 250 to prepare for another INPUT from you.

```
10 POKE 51,255:POKE 52,19:POKE
   55,255:POKE 56,19:CLR
20 FOR Z=0 TO 1240:POKE 5120+
   Z*2,PEEK(32768+Z):POKE 5121+
   Z*2,PEEK(32768+Z):NEXT Z
30 A$="":PRINT "ENTER STRING":
   INPUT A$:IF A$="" THEN 30
40 POKE 36869,253:POKE 36867,255
50 PRINT "";A$
60 GET Z$:IF Z$="" THEN 60
70 POKE 36869,240:POKE 36867,174:
   GOTO 30
```

It is very easy to double the height of the character set on the Vic 20.

The Vic program above sets up the double height letters in Lines 10 and 20 with a series of POKE commands. Because this is a standard feature on the Vic, the computer can just PRINT the string—unlike the other computers' programs, it does not have to split up the string and PRINT one letter at a time.

For this reason, the rest of the program is quite simple: Line 30 lets you enter a string, Line 40 calls the double height characters, Line 50 clears the screen and PRINTs your string, Line 60 stops the computer until you press a key, and Line 70 restores the normal character set, then sends the computer back to Line 30 to let you enter another string.

```
10 LL=40:MODE1
20 PRINT':INPUT"DOUBLE (H)EIGHT AND/OR
   DOUBLE (W)IDTH",A$
30 CLS
40 W=INSTR(A$,"W"):H=INSTR(A$,"H")
50 INPUT"ENTER YOUR WORDS NOW",A$
60 FOR Q=1 TO LENA$
70 C=ASC(MID$(A$,Q,1))-32:IF C<0 OR
   C>95 THEN 170
80 PROCSET
90 IF W<>0 THEN PROCDW
100 IF H<>0 THEN PROCDH
110 IF W=0 THEN 150
120 VDU 224,226
130 IF H<>0 THEN VDU10,8,8,225,
   227,11
140 GOTO 170
150 VDU 224
160 IF H<>0 THEN VDU10,8,225,11
170 IF H<>0 AND Q MOD (LL/2-
   LL/2*(W=0))=0 THEN PRINT
180 NEXT
190 GOTO 20
200 DEF PROCSET
210 FOR T=0 TO 7
220 ?(T+&C24)=?(T+C*8+&C000)
230 ?(T+&C00)=?(T+C*8+&C000)
240 NEXT:ENDPROC
250 DEF PROCDH
260 FOR T=7 TO 0 STEP-1
270 IF W<>0 THEN X=&C00 ELSE
   X=&C24:GOTO 300
280 ?(T*2+&C10)=?(T+X+16)
290 ?(T*2+&C11)=?(T+X+16)
300 ?(T*2+&C00)=?(T+X)
310 ?(T*2+&C01)=?(T+X)
320 NEXT:ENDPROC
330 DEF PROCDW
340 FOR P=0 TO 7
350 ?(P+&C00)=0:?(P+&C10)=0
360 FOR T=7 TO 0 STEP -1
370 ?(P+&C00-16*(T<4))=(?(P+
   &C00-16*(T<4)))*4
380 ?(P+&C00-16*(T<4))=?(P+
   &C00-16*(T<4))-3*((?(P+&C24)
   AND 2^T)<>0)
390 NEXT:NEXT:ENDPROC
```

Lines 10 to 20 set up the computer by putting it into MODE 1, letting you choose between double height letters, double width letters, or double height and width letters, and then clearing the screen.

Line 40 checks the string you have just entered to see whether or not it contains either of the width or height instructions W, or H, or both. If it does, the relevant variable, or variables, are altered.

Then the computer lets you enter your text, which is then stored in A$. The main loop of the program starts at Line 60. By setting this as FOR Q=1 TO LENA$, the computer executes the loop once for every letter in the string you entered.

Using the function ASC, and the string function MID$, Line 7Ø sets a variable, C, equal to the character number of the current letter in the string. The IF . . . THEN condition in this line also checks to see whether the character is valid (between ASCII codes 32 and 127), and if not, the computer jumps to Line 17Ø.

Then, in Line 8Ø, the computer calls PROCSET. This PROCedure uses the ? command both to POKE and to PEEK. Where the sign ? appears before an = sign, it is used to POKE a value into memory. The value it POKEs in is found by PEEKing the number in the second part of the line, in which the ? sign stands for PEEK. This method is used throughout the program.

Lines 21Ø to 24Ø use a FOR . . . NEXT loop to POKE the eight bytes of the current character into two UDGs. Then the PROCedure ends, and the computer returns to Line 9Ø. There, the computer checks to see whether you want double width characters. If you do, it calls PROCDW at Line 33Ø.

DOUBLING UP

This PROCedure splits each byte up into bits, and fill up two UDGs, one byte at a time. As it takes each bit, this is doubled and put into two bits of the UDGs. So 1 gives 11 and Ø gives ØØ. So each bit of the original character is used twice. Once the computer has done this eight times, it has two UDGs, which, when put side by side, form the double-width version of the current character. For example, the byte (in binary) 11ØØ11ØØ would be changed to the two bytes 1111ØØØØ and 1111ØØØØ.

If you select double height, the computer calls PROCDH. The second Line of this routine at Line 27Ø checks to see whether you have already expanded the width of the character—of course, if you have, the computer has to double the height of two UDGs, not just one. If you do not want double width as well as double height characters, the computer jumps to Line 3ØØ, missing out two lines which each set up a UDG.

The Lines 28Ø to 31Ø each set up a UDG equal to eight bytes. Each line POKEs two bytes from memory into the UDG for every one byte of the original character. This is similar to the routine in PROCDW except, instead of doubling each character in width, it doubles the height.

The computer then returns from this PROCedure, and starts the printing routine. This starts off by finding out whether the computer has to PRINT two characters across for every letter (which happens when you have double width characters).

Lines 12Ø to 14Ø are the ones which PRINT the letters in double width. The first line PRINTs two UDGs using a VDU statement. Then, if you choose double height as well as double width, the computer PRINTs two more UDGs, underneath the two it has just PRINTed.

The routine starting at Line 15Ø puts a UDG on the screen, and, if H doesn't equal Ø (in other words, if you do want double height), puts another directly underneath it.

Line 17Ø PRINTs a blank line if 'double height mode' is being used, to make the text easier to read. But it also checks to see whether it is at the end of a line, otherwise each letter would be separated by a line space.

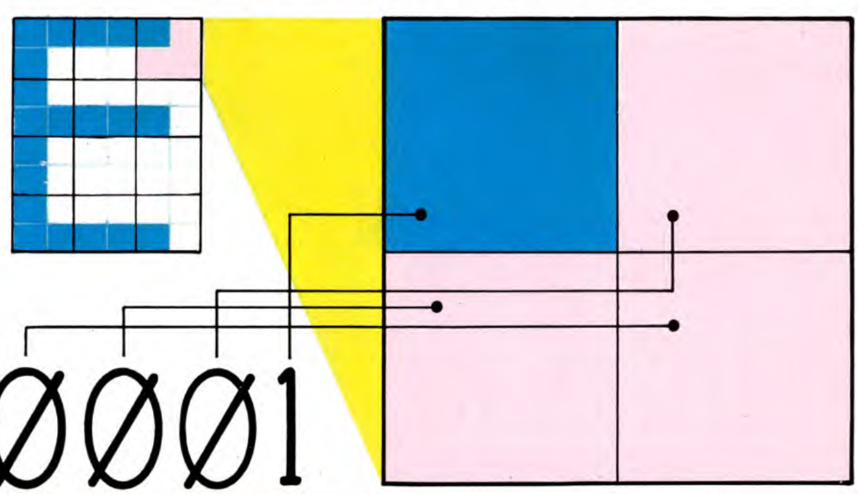
The NEXT in Line 18Ø ends the main loop, sending the computer back to run through it again if there are more letters in the string, or letting it continue to Line 19Ø if there are not.

BLOCK GRAPHIC LETTERS

As well as just expanding your computer's standard characters, you can also create your own designs. This is especially useful on the Dragon and Tandy which do not give you access to the ROM, so you *have* to design your own. Here is a program which uses the block graphics on your computer to build up into large letters.

```
10 DATA "▛▀▌","▛▀▖",
   "▛▀▌","▛▀▖"
20 DATA "▌□▌","▙▄▘",
   "▌□□","▌□▌"
30 DATA "▛▀▌","▌□▌",
   "▌□□","▌□▌"
40 DATA "▌□▌","▙▄▘",
   "▙▄▌","▙▄▘"
50 DATA "▛▀▘","▛▀▘",
   "▛▀▌","▌□▌"
60 DATA "▙▄▖","▌□□",
   "▌□□","▙▄▌"
70 DATA "▌□□","▛▀▘",
   "▌▗▖","▌□▌"
80 DATA "▙▄▖","▌□□",
   "▙▄▌","▌□▌"
90 DATA "▀▛▘","□▜▘",
   "▌▗▘","▌□□"
100 DATA "□▌□","□▐□",
   "▙▘□","▌□□"
110 DATA "□▌□","□▐□",
   "▛▖□","▌□□"
120 DATA "▄▙▖","▄▟□",
   "▌▜▖","▙▄▖"
130 DATA "▙▗▌","▙□▌",
   "▛▀▌","▛▀▖"
140 DATA "▌▌▌","▛▖▌",
   "▌□▌","▌□▌"
150 DATA "▌□▌","▌▚▌",
   "▌□▌","▛▀▘"
160 DATA "▌□▌","▌□▌",
   "▙▄▌","▌□□"
170 DATA "▛▀▌","▛▀▖",
   "▛▀▘,"▀▛▘"
180 DATA "▌□▌","▌□▌",
   "▙▄▖","□▌□"
190 DATA "▌▖▌","▛▜▘",
   "□□▌","□▌□"
200 DATA "▙▟▌","▌□▌",
   "▙▄▌","□▌□"
210 DATA "▌□▌","▌□▌",
   "▌□▌","▌□▌"
220 DATA "▌□▌","▌□▌",
   "▌□▌","▝▞□"
230 DATA "▌□▌","▌□▌",
   "▌▌▌","▗▚□"
240 DATA "▙▄▌","▝▞□",
```

How the blocks are defined on the Acorn computers



```
    "▙▙▐","▐□▐"
250 DATA "▐□▐","▀▀▐"
260 DATA "▙▄▐","□▗▛"
270 DATA "□▐□","▗▛□"
280 DATA "□▐□","▙▄▗"
290 POKE 23658,8: DIM a$(27,3): DIM
    b$(27,3): DIM c$(27,3): DIM d$(27,3)
300 FOR j=0 TO 21 STEP 4
310 FOR i=1 TO 4: READ a$(i+j):
    NEXT i
320 FOR i=1 TO 4: READ b$(i+j):
    NEXT i
330 FOR i=1 TO 4: READ c$(i+j):
    NEXT i
340 FOR i=1 TO 4: READ d$(i+j):
    NEXT i
350 NEXT j
360 FOR i=25 TO 26: READ a$(i):
    NEXT i
370 FOR i=25 TO 26: READ b$(i):
    NEXT i
380 FOR i=25 TO 26: READ c$(i):
    NEXT i
390 FOR i=25 TO 26: READ d$(i):
    NEXT i
400 INPUT "ENTER up to 10 letters", LINE t$:
    IF LEN t$>10 THEN LET t$=t$ (TO 10)
405 IF LEN T$=0 THEN GOTO 400
410 LET s$="": FOR i=1 TO LEN t$: IF
    CODE t$(i)<65 OR CODE t$(i)>90 THEN
    LET t$(i)=CHR$ 91
420 LET s$=s$+a$(CODE t$(i)-64):
    NEXT i
430 PRINT s$
440 LET s$="": FOR i=1 TO LEN t$
450 LET s$=s$+b$(CODE t$(i)-64):NEXT i
460 PRINT s$
470 LET s$="": FOR i=1 TO LEN t$
480 LET s$=s$+c$(CODE t$(i)-64):
    NEXT i
490 PRINT s$
500 LET s$="": FOR i=1 TO LEN t$
510 LET s$=s$+d$(CODE t$(i)-64):NEXT i
520 PRINT s$
530 PRINT : GOTO 400
```

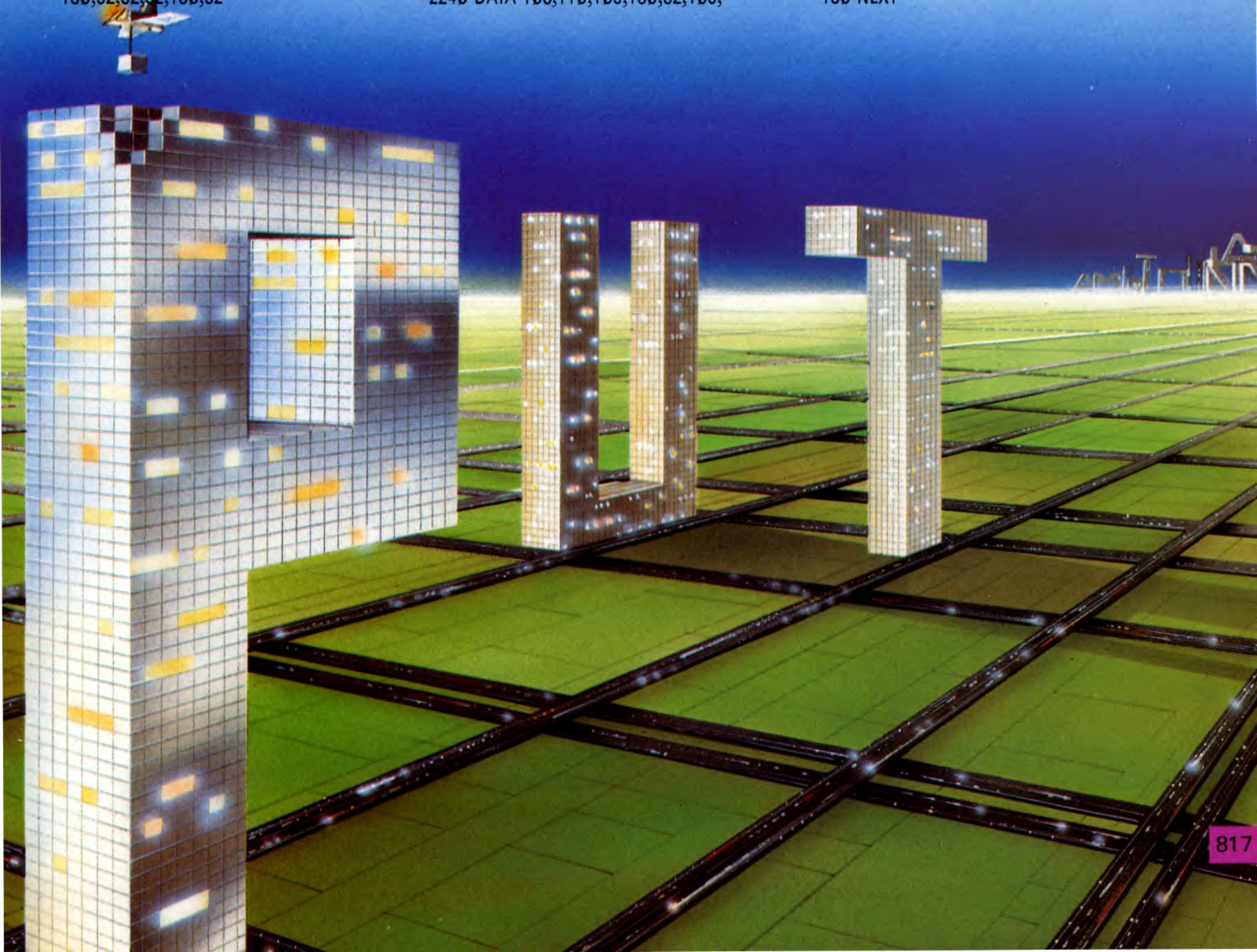
For the Vic, change the 9s in Line 170 to 5s.

```
100 DIMA(78),B(78),C(78),D(78)
120 FOR I=1TO78:READA(I):NEXT
130 FOR I=1TO78:READB(I):NEXT
140 FOR I=1TO78:READC(I):NEXT
150 FOR I=1TO78:READD(I):NEXT
170 T$="":PRINT"▣▣ENTER UP
    TO 9 LETTERS":INPUTT$:IFLEN(T$)
    >9THEN170
180 IFLEN(T$)=0THEN170
185 FORX=1TO4
190 FORY=1TOLEN(T$)
200 A$=MID$(T$,Y,1)
210 A=ASC(A$):A=A-64
220 B=(A-1)*3
225 ONXGOTO230,240,250,260
226 GOTO170
230 FORR=B+1TOB+3:PRINTCHR$
    (A(R));:NEXT:PRINT"□";:NEXT:
    PRINT:NEXT
240 FORR=B+1TOB+3:PRINTCHR$
    (B(R));:NEXT:PRINT"□";:NEXT:
    PRINT:NEXT
250 FORR=B+1TOB+3:PRINTCHR$
    (C(R));:NEXT:PRINT"□";:NEXT:
    PRINT:NEXT
260 FORR=B+1TOB+3:PRINTCHR$
```



```
   (D(R));:NEXT:PRINT"□";:NEXT:
   PRINT:NEXT:GOTO17Ø
2ØØØ DATA 111,183,112,111,183,1Ø9,
   111,183,112,111,183,1Ø9
2Ø1Ø DATA 111,183,183,111,183,183,
   111,183,112,18Ø,32,17Ø
2Ø2Ø DATA 112,183,32,32,112,183,17Ø,
   32,11Ø,18Ø,32,32
2Ø3Ø DATA 1Ø8,32,186,1Ø8,32,17Ø,111,
   183,112,111,183,112
2Ø4Ø DATA 111,183,112,111,183,112,
   111,183,112,183,111,32
2Ø5Ø DATA 18Ø,32,17Ø,18Ø,32,17Ø,18Ø,
   32,17Ø,18Ø,17Ø,32
2Ø6Ø DATA 18Ø,32,17Ø,183,183,18Ø
2Ø65 DATA 18Ø,32,17Ø,18Ø,32,17Ø,18Ø,
   32,32,18Ø,32,17Ø
2Ø7Ø DATA 1Ø8,175,32,1Ø8,175,32,18Ø,
   32,32,18Ø,32,17Ø
2Ø8Ø DATA 17Ø,32,32,32,17Ø,32,17Ø,
   11Ø,32,18Ø,32,32
2Ø9Ø DATA 18Ø,184,17Ø,18Ø,1Ø9,17Ø,
   18Ø,32,17Ø,1Ø8,175,186
21ØØ DATA 18Ø,32,17Ø,1Ø8,175,186,
   18Ø,32,32,32,18Ø,32
211Ø DATA 18Ø,32,17Ø,18Ø,32,17Ø,18Ø,
   32,17Ø,1Ø9,11Ø,32
212Ø DATA 1Ø9,175,11Ø,32,11Ø,32
213Ø DATA 111,183,112,111,183,112,
   18Ø,32,32,18Ø,32,17Ø
214Ø DATA 18Ø,32,32,18Ø,32,32,18Ø,
   32,112,111,183,112
215Ø DATA 17Ø,32,32,32,17Ø,32,17Ø,
   1Ø9,32,18Ø,32,32
216Ø DATA 18Ø,32,17Ø,18Ø,32,112,18Ø,
   32,17Ø,18Ø,32,32
217Ø DATA 18Ø,1Ø9,11Ø,18Ø,1Ø9,32,
   183,183,112,32,18Ø,32
218Ø DATA 18Ø,32,17Ø,18Ø,32,17Ø,18Ø,
   98,17Ø,11Ø,1Ø9,32
219Ø DATA 32,125,32,11Ø,32,32
22ØØ DATA 18Ø,32,17Ø,1Ø8,175,11Ø,
   1Ø8,175,186,1Ø8,175,11Ø
221Ø DATA 1Ø8,175,175,18Ø,32,32,1Ø8,
   175,186,18Ø,32,17Ø
222Ø DATA 186,175,32,1Ø8,186,32,17Ø,
   32,1Ø9,1Ø8,175,175
223Ø DATA 18Ø,32,17Ø,18Ø,32,17Ø,1Ø8,
   175,186,18Ø,32,32
224Ø DATA 1Ø8,11Ø,1Ø9,18Ø,32,1Ø9,
   1Ø8,175,186,32,18Ø,32
225Ø DATA 1Ø8,175,186,1Ø9,175,11Ø,
   1Ø8,177,186,18Ø,17Ø,32
226Ø DATA 32,125,32,1Ø8,175,
   32
```

```
1Ø MODE1:D=128Ø/4Ø/2
2Ø DIM A(311):FOR T=Ø TO 311:
   READ A(T):NEXT
3Ø VDU23,224,24Ø,24Ø,24Ø,24Ø,Ø,Ø,Ø,Ø
4Ø INPUT'"A WORD NOT MORE THAN 1Ø
   CHARS LONG ",B$
5Ø IF LEN(B$)>1Ø THEN CLS:PRINT"TOO
   LONG":GOTO4Ø
6Ø PX=Ø:PY=7ØØ
7Ø FOR P=1 TO LEN(B$):A$=
   MID$(B$,P,1):PROCCHAR:NEXT
8Ø G=GET:CLS:GOTO 4Ø
9Ø DEF PROCBLOCK
1ØØ VDU 5
11Ø FOR T=Ø TO 3
12Ø IF B AND 2∧T THEN MOVE Ø+(T AND
   1)*D,Ø-(T AND 2)*D/2:VDU 224
13Ø NEXT
```





Block letters on the Dragon



... and on the Spectrum

```
140 VDU 4
150 ENDPROC
160 DEF PROCCHAR
170 C=ASC(A$)-65
180 IF C=-33 THEN 220
190 IF C<Ø OR C>25 THEN ENDPROC
200 FOR X=Ø TO 2:FOR Y=Ø TO
   3:B=A(X+Y*3+C*12)
210 VDU 29,PX+X*D*2;PY-Y*D*2;:
   PROCBLOCK:NEXT:NEXT
220 PX=PX+128
230 ENDPROC
240 DATA 7,3,5,5,Ø,5,7,3,5,5,Ø,5,7,3,4,13,12,
   1,5,Ø,5,13,12,1
250 DATA 7,3,5,5,Ø,Ø,5,Ø,Ø,13,12,5,7,3,4,5,Ø,
   5,5,Ø,5,13,12,1
260 DATA 7,3,1,13,12,4,5,Ø,Ø,13,12,4,7,3,1,5,
   Ø,Ø,7,3,1,5,Ø,Ø
270 DATA 7,3,5,5,Ø,Ø,5,8,4,13,12,5,5,Ø,5,13,
   12,5,5,Ø,5,5,Ø,5
280 DATA 3,7,1,Ø,5,Ø,Ø,5,Ø,12,13,4,Ø,11,1,Ø,
   1Ø,Ø,Ø,1Ø,Ø,13,14,Ø
290 DATA 5,8,1,13,1,Ø,7,4,Ø,5,2,4,5,Ø,Ø,5,Ø,
   Ø,5,Ø,Ø,13,12,4
300 DATA 13,8,5,5,5,5,5,Ø,5,5,Ø,5,13,Ø,5,7,4,
   5,5,9,5,5,2,5
310 DATA 7,3,5,5,Ø,5,5,Ø,5,13,12,5,7,3,5,5,Ø,
   5,7,3,1,5,Ø,Ø
320 DATA 7,3,5,5,Ø,5,5,4,5,13,14,5,7,3,5,5,Ø,
   5,7,11,1,5,Ø,5
330 DATA 7,3,5,13,12,4,Ø,Ø,5,13,12,5,3,7,1,Ø,
   5,Ø,Ø,5,Ø,Ø,5,Ø
340 DATA 5,Ø,5,5,Ø,5,5,Ø,5,13,12,5,5,Ø,5,5,Ø,
   5,5,Ø,5,2,6,Ø
350 DATA 5,Ø,5,5,Ø,5,5,5,5,13,13,5,5,Ø,5,2,6,
   Ø,8,9,Ø,5,Ø,5
360 DATA 5,Ø,5,13,12,5,Ø,5,Ø,Ø,5,Ø,3,3,5,Ø,8,
   1,8,1,Ø,13,12,4
```

```
10 DIM L(2,3,25)
20 FOR J=ØTO25:FOR K=ØTO3:
   FOR L=ØTO2:READ L(L,K,J):
   NEXT L,K,J
30 CLSØ
40 PRINT@48Ø," INPUT WORD ?";B$;
50 A$=INKEY$:IF (A$<"A" OR A$>"Z")
   AND A$<>CHR$(8) AND
   A$<>CHR$(13) AND A$<>"□"
   THEN 5Ø
60 IF A$=CHR$(13) THEN 11Ø
70 IF A$=CHR$(8) AND B$="" THEN 5Ø
80 IF A$=CHR$(8) THEN B$=LEFT$
   (B$,LEN(B$)-1):GOTO3Ø
90 IF LEN(B$)>9 THEN 5Ø
100 B$=B$+A$:GOTO3Ø
110 IF B$="" THEN CLS:END
120 CLSØ:PRINT@48Ø," COLOUR (1-8)?";
130 A$=INKEY$:IF A$<"1" OR A$>"8"
   THEN 13Ø
140 CLSØ:CL=VAL(A$)
150 FORY=ØTO3:FORC=1TOLEN(B$):
   FORX=ØTO2
160 IF MID$(B$,C,1)="□" THEN PRINT
   CHR$(128);:GOTO18Ø
170 PRINTCHR$(84+CL*16+L(X,Y,
   ASC(MID$(B$,C,1))-65));
180 NEXTX,C:PRINTSTRING$(32-
   POS(Ø),128);:NEXT Y
190 B$="":GOTO4Ø
1000 DATA 42,40,38,38,28,38,42,40,38,
   38,28,38,42,40,30,39,31,36,38,28,
   38,39,31,36
1010 DATA 42,40,38,38,28,28,38,28,28,39,31,
   38,42,40,30,38,28,38,38,28,38,39,31,36
1020 DATA 42,40,36,39,31,30,38,28,28,39,31,
   30,42,40,36,38,28,28,42,40,36,38,28,28
1030 DATA 42,40,38,38,28,28,38,29,30,39,31,
   38,38,28,38,39,31,38,38,28,38,38,28,38
1040 DATA 40,42,36,28,38,28,28,38,28,31,39,
   30,28,41,36,28,33,28,28,33,28,39,35,28
1050 DATA 38,29,36,39,36,28,42,30,28,38,32,
   30,38,28,28,38,28,28,38,28,28,39,31,30
1060 DATA 39,29,38,38,38,38,38,28,38,38,28,
   38,39,28,38,42,30,38,38,37,38,38,32,38
1070 DATA 42,40,38,38,28,38,38,28,38,39,31,
   38,42,40,38,38,28,38,42,40,36,38,28,28
1080 DATA 42,40,38,38,28,38,38,30,38,39,35,
   38,42,40,38,38,28,38,42,41,36,38,28,38
1090 DATA 42,40,38,39,31,30,28,28,38,39,31,
   38,40,42,36,28,38,28,28,38,28,28,38,28
1100 DATA 38,28,38,38,28,38,38,28,38,39,31,
   38,38,28,38,38,28,38,38,28,38,32,34,28
1110 DATA 38,28,38,38,28,38,38,38,38,39,39,
   38,38,28,38,32,34,28,29,37,28,38,28,38
1120 DATA 38,28,38,39,31,38,28,38,28,28,38,
   28,40,40,38,28,29,36,29,36,28,39,31,30
```

Each of these programs works in a similar way. They begin by DIMensioning arrays for the letters. The Spectrum also POKEs a system variable to turn CAPS LOCK on. The Spectrum program uses four arrays—one for each row of the letters, while the other computers' versions just use one array.

The delay you probably meet when you RUN the program is caused by the computer READing the DATA into the array, or arrays. The Spectrum version uses Lines 29Ø to 39Ø to do this, while the Acorn, Dragon and Tandy programs do it in just one Line (Line 2Ø).

Both the Acorn computers then set up a UDG, since only the BBC has block graphics, and it can only use these in MODE 7. The UDG is used to represent each possible graphic, in MODE 1, by PRINTing it several times in slightly different positions so that it forms the correct character.

The computers now start the main section of the program by letting you enter a word. The Spectrum, Commodore 64, Acorn, Dragon and Tandy computers let you have up to a maximum of 10 letters. The Vic gives you just 5 as it has a very small screen.

Each computer performs various checks to make sure that you have typed in a valid string, and then begins a FOR ... NEXT loop to PRINT the large letter. This loop is the same as the one in the last program in this article, in that it carries on until the computer has run through it once for every character in the string.

The Spectrum actually uses four of these loops, one for each row of block graphics as the large letters are all four characters tall. The loop adds a group of three characters to s$, for each letter in the string. When there are no more letters in the string, the computer PRINTs s$, which will be the next line of graphics characters.

The string which is added to s$ every time is changed from loop to loop. As you can see from lines 42Ø, 45Ø, 48Ø, and 51Ø, the first string which is added is a$, then b$, c$, and finally d$. Each of these is in fact an array, and the calculation in brackets after it determines which element of the array is added to s$.

Once the Spectrum has PRINTed all four rows, it goes to Line 4ØØ to let you enter another word.

The Commodore programs start the main loop in Line 185. In Line 2ØØ the computer sets A$ equal to the current letter from your string. Then the variable A is set to the ASCII code of this letter, less 64. B is a variable which the computer uses to PRINT the characters later on in the program, and is set up in Line 22Ø.

Line 225 uses the ON ... GOTO facility to send the computer to one of four lines, each of which PRINTs one of the rows of a letter. In fact the line reads ONXGOTO ..., where X is the control variable of the loop in Line 185.

When the computer jumps to one of these four lines, it starts yet another FOR ... NEXT loop, which it uses to PRINT out the three characters in each row of the large letter. The new loop is from B+1 to B+3. B was set equal to a value calculated from the current character's ASCII code, and is used to call up the correct element of the array.

Once the computer has PRINTed each of the three characters, it PRINTs a space (to separate this letter from the next) and goes on to the NEXT value of Y (this makes the computer PRINT the next row of the next letter in your string). As soon as the three characters in the top row of all the letters in your string have been PRINTed, the computer goes to the NEXT value of X. This means that the computer does the same process again, but for the second row, and then the third and fourth rows, of characters. As there are only four rows of characters for each letter, the computer returns to let you enter another string.

Once the computer has set up two variables, PX and PY, it starts its first loop. The loop, in Line 7Ø, simply calls PROCCHAR once for every character in the string you entered.

PROCCHAR first sets the variable C equal to 65 less than the ASCII code of the current letter. If C equals −33 (in other words if the actual ASCII code of the letter is 32, the code for a space) then the computer jumps to Line 22Ø, which adds 128 to the variable PX to move the PRINT position one big-letter-space to the right—effectively a space.

Line 19Ø makes sure that the current letter is between A and Z. If it is not, the computer returns from the PROCedure to Line 7Ø.

The two loops in Line 2ØØ, make the computer run through Lines 2ØØ and 21Ø once for every character space that each letter takes up (or twelve times in all). These two lines change the cursor's origin (the position Ø,Ø) to the start of the next letter's position, and then call PROCBLOCK.

PROCBLOCK, which starts in Line 9Ø, uses the graphics cursor to PRINT the UDG up to four times in different places, so that the result is the same as a block graphic from one of the other computers. The complicated-looking Line 12Ø works out whether the computer needs to PRINT the UDG for that value of T. T is the control variable of the FOR ... NEXT loop in this PROCedure, and goes from Ø up to 3. The calculation uses logical arithmetic (the function AND), and an IF ... THEN condition. If it does need to PRINT a UDG for any value of T, the computer MOVEs the graphics cursor to the correct position.

The DATA held in the array comes in at this point. Each piece of DATA represents a four digit binary number—so the maximum is 1111 in binary, which equals 15. Each bit of the binary number refers to one of the four quarters of a graphics square, as shown in the diagram on page 819. (Block graphics are just a graphics character split into quarters, and with one or more of the quarters filled in).

When the computer finds a letter in the string you entered, it takes each character square for that letter and converts the corresponding number from the DATA into binary. This binary number is then used in Line 12Ø to determine which, if any, of the four quarters are filled in.

After the computer has finished each character, it returns from PROCBLOCK to PROCCHAR, and finds the next character for the current letter. When all twelve character squares of each letter have been PRINTed by PROCBLOCK, the computer increases the variable PX by 128 to move the PRINT position one space on, ready for the next letter, and so on.

After every letter has been PRINTed from the string you entered, the computer goes back to Line 8Ø, where it waits for you to press a key before clearing the screen, and starting again.

As soon as you have entered your text and pressed ENTER, the computer jumps to Line 11Ø. You can then choose which colour you want the computer to PRINT your text in. Lines 13Ø and 14Ø deal with setting the colour, and then the computer starts three FOR ... NEXT loops.

Line 16Ø PRINTs a black space whenever a space is the next character in the string you entered. After PRINTing this, the computer jumps to Line 18Ø, missing out Line 17Ø.

Line 17Ø, although it looks extremely complicated, just PRINTs the next block graphic character for the current letter. The hard part is the calculation in brackets which works out the CHR$ number. The first bit of it adds 84 to the colour number*16, and adds this to the relevant item of DATA from the array. The L(X,Y,ASC(MID$(B$,C,1))−65)) determines which element of the array L is used in the calculations, as explained below.

The DATA in the array is the CHR$ numbers of the various different block graphics (the basic, black and green, ones) minus 1ØØ. They are arranged so that the twelve numbers for each letter (four rows of three characters) follow on, so the thirteenth number in the array refers to the letter B, the twenty fifth number is the first character for C, and so on.

When the computer works out which number in the array to read, Line 17Ø uses the function ASC to give the character number of the letter from your string, and it uses MID$ to work out which letter in your string it is dealing with. The three numbers in brackets after the MID$ refer to which string is being 'cut up' (here B$); how many characters into the string the first letter you want is (here C, the control variable from one of the FOR ... NEXT loops set up in Line 15Ø); and how many characters from this point (here one).

The semi-colon after the calculations in Line 17Ø prevent the next PRINT statement from starting on a new line, so that all the letters appear on the same line.

Line 18Ø first does two NEXT statements, to go onto the next value for X (which governs which of the three characters in each row of the current character is PRINTed). The second NEXT, for the next value for C is simply added on to the first by putting ,C after the X.

The second part of Line 18Ø PRINTs enough black spaces to move the PRINT position onto the next line, so that the computer can PRINT the next row of graphics—there are four rows in all. When the last loop started in Line 15Ø has finished, your word will be on the screen in your chosen colour, and in large type. Line 19Ø makes the computer wait for you to press a key, after which the computer jumps back to Line 4Ø to let you INPUT another word.

When you have had enough of the program, or want to return to BASIC, you can either enter a 'null' string (press ENTER before typing any letters when the computer expects your word), or press BREAK.

The programs in this article have showed just two possible ways of creating impressive type faces to brighten up your programs' title pages. The next article will show another way of creating a very different sort of typeface, and will explain how you can take letters created by all of the methods and use them in your own programs.

SETTING UP A DISK DRIVE

Unlike a tape recorder, a disk drive can be complicated to set up and use. This article helps you to avoid the common mistakes, and explains the BASIC you need.

If you decide that it's time for something a bit quicker and easier than tape storage, disk units have a lot to offer for the extra cost. But setting up a disk/computer combination is rather more complicated than using a tape machine, and you have to become accustomed to using new commands to control it.

This article deals with the procedure for connecting and using the most popular disk drive units for each computer. The Spectrum instructions are an exception, since they cover the Sinclair Microdrive which is the most practical and popular alternative to tape, rather than disk units for this machine, which are far less common and not specifically designed for the job. The Commodore 64 and Vic 20 notes apply to the CBM 1540 and 1541 disk units. The BBC notes apply equally to all drives—but as interfaces are still being developed for the Electron, this is not covered, and the Acorn instructions apply only to the BBCB. The Dragon instructions are for Dragon Data's own disk unit.

The first thing you should do when you get your disk drive is check to make sure that there is no transit packing inside it—some manufacturers put a piece of cardboard inside to protect the head. If your drive has one of these, you should take it out—but keep it, just in case you want to move your drive around in the future. Then you are all set to plug in your drive to your computer.

Although you can get interfaces to connect various models of disk drive to the Spectrum, this is an expensive option and the most popular alternative is Sinclair's own Microdrive based on a miniature tape cartridge. The advantages of it over a 'real' disk drive are mainly in terms of price, but also, it is easy to connect the drive to your Spectrum. And several Microdrives (a maximum of eight) can be linked to increase your storage, although you only access one at a time.

However, you do need to have a Sinclair Interface 1 as well as the Microdrive unit itself, which effectively doubles the cost of the first drive (even if you have more than one unit, you only need one Interface.

The Microdrive comes with a short ribbon cable, and you should plug this into the drive and into the Interface 1. Each Microdrive also has a separate hard connector used for linking to another drive in daisy chain form.

You should connect the Interface 1 to the Spectrum, of course, too. Sinclair suggest that you do this permanently—by fastening the screws on the interface to the computer after plugging into the edge connector. You may find it better not to do this, though, as some software does not work if the interface is plugged in and it is useful to be able to remove it easily.

Once you have connected the interface to your Spectrum, and the Microdrive to the interface, you can turn on. The Microdrives are powered from the computer, however many you have, so you just turn on the computer in the usual way. But do not power up with a cartridge in the slot, you may corrupt the information stored on it.

To check whether or not you have correctly plugged everything in, you can insert a cartridge into the slot in the Microdrive, and then type RUN and press ENTER (if you have been using the computer, you must NEW it first). The Microdrive should now come to life—the red light on its front will light up, and you should hear a whirring sound as the tape inside spins round. If this happens, your drive is correctly connected. If not, then you should check your connections and try again.

If you have more than one Microdrive connected, this test will only access the first of



Microtip

Switching on

Be very careful when switching your disk drive on or off. Always make sure that there is no disk in it before you do so. Disks are very sensitive and it is possible to erase part of the directory if the chips in the disk drive are powered up—or down—when a disk is inside. The directory is on the part of the disk where the head usually rests so it is particularly vulnerable. And if the directory is corrupted it is impossible to retrieve the information that is on the rest of the disk. The software has no way to locate where the files are stored.

- INTERFACING YOUR MICRODRIVE
- DAISY CHAINING
- CALLING THE CATALOGUE OR DIRECTORY
- DANGERS TO DISKS
- ACORN AND VIC MODIFICATIONS
- FORMATTING DISKS
- HANDLING FILENAMES
- USING THE SOFTWARE

them. To test the others, type CAT, followed by the number of the Microdrive (counting away from the interface).

Plug your Commodore disk drive into the mains, as it uses its own power supply, and turn on. You should then see a green light, which simply shows when the power is on, and the red light should flash on and then off. If this does not happen, the power is not reaching it—or your drive is faulty.



If you have more than one peripheral connected to your computer, for example, a disk drive and a printer, then you should take care how you switch each one on. The Commodore 64 in particular is rather sensitive as to the order in which you do it—you may even destroy the I/O chip. Contrary to what is said in some early Commodore peripherals manuals, you should always switch the computer on first, followed by the disk drive (followed by any other disk drive if you daisy-chain them), followed by the printer. It does not matter, though, what order you connect them up in—you can either join the printer to the disk drive, and the drive to the computer, or join the drive to the printer, and then the printer to the computer.

Before a Commodore 1541 disk drive can be used with a Vic 20, the speed at which the unit runs. may have to be adjusted. To do this, you will have to consult your manual for the appropriate commands.

The instruction to set up the disk drive must be entered before you enter any other instructions. If the disk drive is being called from a program, the instruction is normally put at the very beginning of the program. Once you enter it, the command stays in force until power is switched off.

Probably the most important point to note is that before you can fit a disk drive to your BBCB, you must first add eleven new chips to the computer. These comprise a 'Disc Filing System ROM' (DFS), of which there are several available, and ten others which provide the interface for the drive unit. These can be fitted for you by a dealer, or bought as a kit to fit yourself—usually a cheaper option.

Once you have fitted these chips into your BBC, you can plug in your disk drive. Its ribbon cable plugs directly into the disk port underneath your computer. Your disk drive may be powered in one of two ways depending on the design—either using the computer's power supply, or from a separate supply. The latter type needs to be plugged into the mains.

If your disk drive relies on the computer for its power, it will probably have a lead for this purpose. If it has, you should plug it into the power supply of your computer—this is situated next to the disk drive port.

Dragon Data disk drives plug into the computer's cartridge port—as does the most popular independent unit. The first thing you plug in, in fact, is a cartridge containing the software needed to use the disk drive. This has a port into which you can then plug the disk drive. Power for the disk drive comes from a separate mains connection.

FORMATTING

Before you can use any disk to store information, you have to set it up—this is called formatting, or initializing a disk. All that this does is to organize the way information is stored on the disk.

When you do this with the Sinclair Microdrive, it detects whether any section of the tape inside the cartridge is not reliable enough to use—this is why, when you FORMAT a cartridge, the amount of storage space will not always be the same. Normal disks are divided into ten or eleven imaginary sections, like slices of cake. While the tracks and sections of a disk do not have actual physical boundaries, at least none which you could see, they are precise limits, which the disk drive recognizes. This is known as soft sectoring, since it is the software which sets up the boundaries.

Here are the commands you can use to format a new disk (or cartridge):

```
FORMAT"M";1;"cartridge name"
```

```
OPEN 1,8,15,"NEW:(diskname), (two letters for identification)"
```

You could also open a logical file, and then use PRINT #, but the above command is simpler. There is an alternative to formatting a disk, which is to INITIALIZE it. This is much more rarely used—generally when you are working on many disks and there is a risk of corruption of existing data. You should normally use the form given above.

The commands you should use with a BBC disk drive to format a disk vary, depending upon which disk ROM you have.

Acorn's own disk system needs a whole program to format a disk, and the program comes with the system.

```
DSKINIT
```

Formatting a disk, or cartridge, actually takes about a minute—again, this depends upon what disk drive and computer you have.

Once you have formatted a disk, you do not need to do so again, although you can. Of course, if you do reformat a disk, it will be wiped clean, and so you lose whatever was stored on it. If you have finished with an entire disk's collection of files, this can be a useful way of erasing every file at one go.

FILENAMES

Everything which you store on either tape or disk has to have a filename. When you SAVE something on tape, you give a name to whatever you are storing simply by putting the name in quotes after the SAVE command. Disk files have to be given a name too.

The Spectrum Microdrives allow you to have a filename up to 10 characters long; the Commodore disk unit has a maximum of 16 characters; the Acorn may support filenames of up to 10 characters, while the Dragon can only have filenames up to 8 characters long.

Filenames form a much more important part of a disk system, since each disk has a directory which is constantly, and automatically, updated by the computer whenever an amendment is made to what is stored on the disk. This is explained in more detail further on in this article.

USING YOUR DISK DRIVE

As with printers, joysticks, light pens, and most other peripherals, the key to using your disk drive is software—in fact, the quality of a disk drive is often judged by the software support that comes with it, rather than by the hardware. This is more logical than it sounds, because it is not really the disk unit that you are going to use, but the software which controls it.

On the Spectrum with Microdrive(s), you do not use a 'general' command to direct output which is the way most computers work with a disk drive, but use a slightly different version of the normal tape SAVE, LOAD, and VERIFY commands. The advantage of this over a general command is that you can use both cassette and Microdrive without having to continually change the 'general' state.

The new LOAD command looks like this:

```
LOAD *"m";1;"filename"
```

Here, the asterisk tells the Spectrum that the command is part of the new extended BASIC present in Interface 1—most of the commands which can be used with the Interface 1, but are also available normally, use this character, while special commands which are unique to the interface do not.

The m in quotes tells the computer that the Microdrive is the peripheral to which the information is being sent, while the 1 inside semi-colons tells it the number of the Microdrive (remember you can have up to 8 in use).

To save information on the Microdrive cartridge, you will also need to use this new SAVE command, which works in the same way:

```
SAVE *"m";1;"filename"
```

You can also use the VERIFY and MERGE commands with the Microdrives, by adding the same set of information after the command (so you would use, for instance, VERIFY *"m";1;"filename").

One important point about LOADing, VERIFYing or MERGEing the programs stored on a Microdrive cartridge is that you must be specific about the filename of any program or block of CODE and give this *exactly* as in the original SAVE command. You cannot simply use the equivalent form of the tape command LOAD"". Luckily, though, you do not need to keep efficient records of what you called your latest program, as you can ask the computer to tell you what is on any one cartridge.

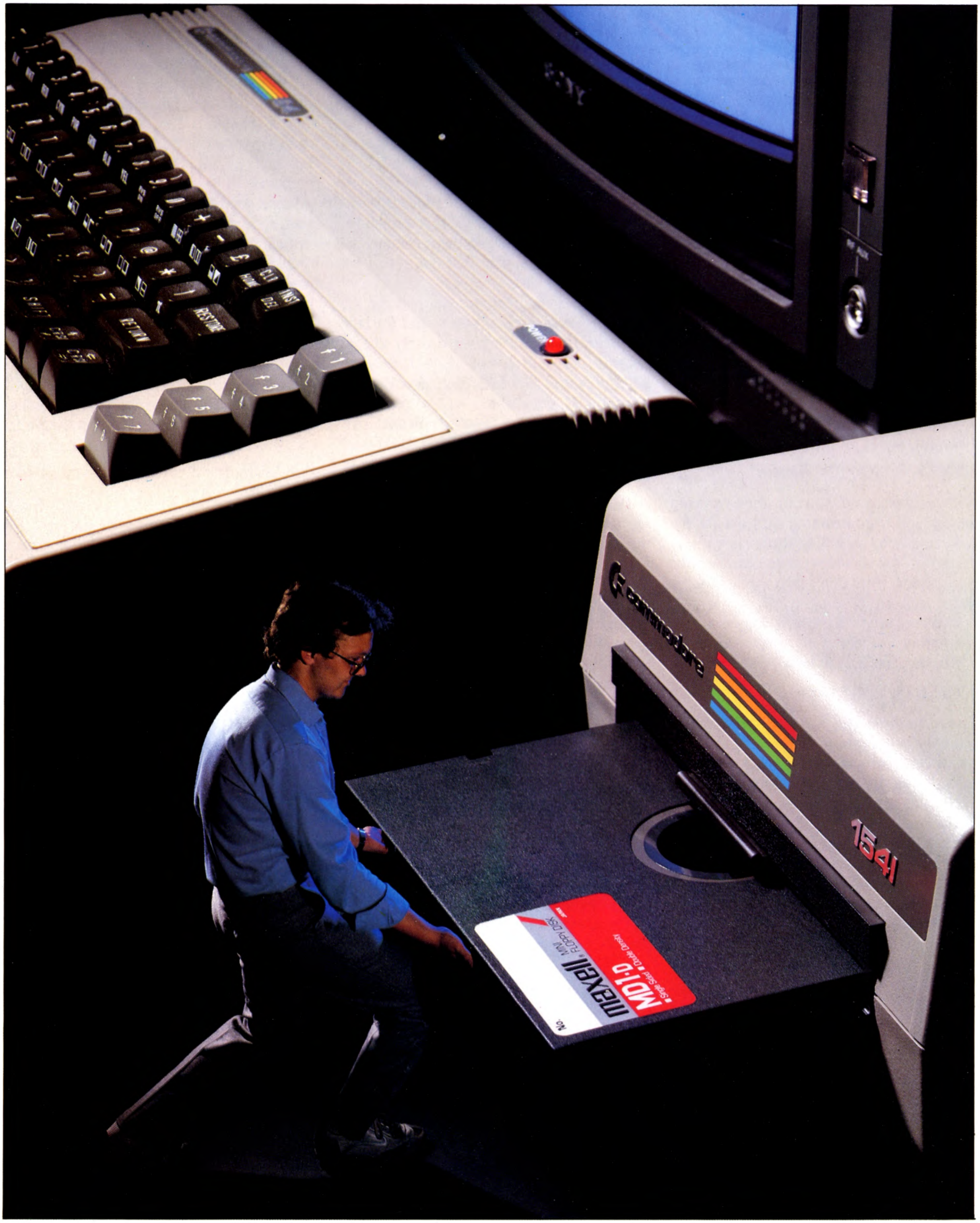
The command you should use for this is CAT followed by the number of the Microdrive that you are using (this will usually be 1). If you type this and press ENTER, the Microdrive's red light will light up, and after a few seconds the CATalogue of what is on the cartridge, together with the amount of storage space remaining, is PRINTed on the screen. When you use this command, the very first thing that is PRINTed up is the name of the cartridge, which you assigned when you FORMATted it.

There is another command, too, which you can use with your Microdrive: ERASE. This does what you would expect it to—erases a file from the Microdrive cartridge. It takes the same form as the LOAD and SAVE commands, except that it does not need the asterisk. So this command:

```
ERASE "m";1;"badprogram"
```

would wipe out a file named badprogram from the cartridge in Microdrive 1.

Unlike cassette based software, you need this ERASE command with Microdrives, since you cannot store your new files over anything else—and you cannot have more than one file with the same name on a cartridge, either. So, if you have finished with any particular file on


commodore
1541
maxell
MD1-D

cartridge, you should erase it to make room for more programs.

The commands you use to access your Commodore disk drive are almost the same as those for tape—the LOAD and SAVE commands simply have an extra ,8 after the filename. So, to SAVE a program named INPUT to disk, you would type in this command:

```
SAVE"INPUT",8
```

There is also an extra command which is very useful—it lets you see what programs are stored on the disk. To do this, enter:

```
LOAD"$",8
```

When the message READY appears, LIST the program, and a list of the files will be PRINTed on the screen.

To erase a file, or a program, from your disk, you should use this command:

```
OPEN 1,8,15,"SCRATCH: filename"
```

With this command, you can delete as many files as you like, assuming they are on the disk, simply by including their name inside the string. You should separate each filename by a comma. (And don't forget to CLOSE the file.)

If you want to erase everything on a disk, you can do so either by reformatting, or initializing, the disk, or with this command:

```
OPEN 1,8,15,"S:*": CLOSE 1
```

This command also shows another point about the disk commands—you can abbreviate the commands inside the string to a single letter. Here, for example, the SCRATCH is reduced to just S and you don't even need to follow it by a full stop.

There may be occasions when you want to rename a file that you have already stored on disk. One way would be to LOAD the file into your computer, erase it from the disk, and then SAVE it again with the new name. But this is very risky because you end up with a situation where the only copy of the program is the one in the computer—and one power cut would lose the lot. So Commodore have provided a feature which does the job for you in a much simpler way—the RENAME command. To rename a file called UDG GENERATOR to one named UDGED, you can use this command:

```
OPEN 1,8,15,"R:UDGED=UDG
   GENERATOR": CLOSE 1
```

As you can see, the first filename in the string is the new one, and the second is the old.

You do not need to tell your BBC that you are using a disk drive, as it detects this automatically if the unit is plugged in. If you want to use a tape recorder while the disk drive is plugged in, you should first enter *TAPE. When you want to return to disk again, you can enter *DISK.

Once you have a formatted disk and your disk drive all plugged in, you probably want to use it to SAVE and LOAD your programs. Unless you just want to use commercial software from disk, your first task is to store something on a disk.

To SAVE a program, you use exactly the same SAVE command as you would for tape (assuming that the computer is ready to pass information to disk—see above). You can LOAD programs back from disk, too, using the normal LOAD command.

There are several new commands, too, provided by both the computer's ROM, and the new Disk Filing System chip that has to be fitted. Many commands work on both tape and disk. Probably the most useful of these is the *CAT command. This PRINTs up a list of all the files stored on the disk in the drive. And although you need to know the filename of something before you can LOAD it, this command means that you do not need to remember the names of all your files, as it only takes a few seconds for the catalogue to be PRINTed on the screen.

You can also use a slightly fuller version of this command—*INFO. This gives more information about each file, but as it works slightly differently on each DFS, you should look at your manual to find out more about it.

You can delete a file that you have finished with by using this command:

```
*DELETE□filename
```

You should note here, that the filename is not in quotes or brackets but it can be, unlike names of programs on tape.

There's also a command that can rename a file which you have already stored on disk:

```
*RENAME□oldname□newname
```

As you can see from this command, you should put the current name of the file after the RENAME, followed by a space, and then the name you want the file to have.

There are other special purpose commands you can use, too, but which of these are available depends upon which Disk Filing System you have. Your DFS manual will list all other new commands you can use, but if *HELP is available with the DFS, it should print out the possible commands for you.

The Dragon commands to SAVE and LOAD programs to and from disk are actually simpler than those for tape—instead of CSAVE, which is the tape command, you just use SAVE"FILENAME". The same also applies to loading programs from disk—you do not need the letter C at the beginning of each command.

When you first turn on, the VERIFY option is set, so that everything you store on disk is verified automatically. You can turn this off by entering, quite simply, VERIFY OFF. If you then want to reinstate it, enter VERIFY ON.

When you SAVE something on to disk, although you just type SAVE"FILENAME" and then press ENTER, the computer adds a suffix to the end of the filename. This can be one of four things—BAS, which signifies a BASIC program; BIN, which signifies a machine code program; BAK, which indicates that the file is a BAcKup of another file on the same disk; or DAT, which indicates that the file is simply DATA—not a program.

When you LOAD a BASIC program from disk into your Dragon, you do not need to type the suffix as well—just the filename which you gave the program will do.

You can see how this works by SAVEing a short program, and then entering:

```
DIR
```

You should then see the filename of your program appear on the screen, with the suffix .BAS. DIR, the directory command, is very useful—what it does is to PRINT onto the screen a list of every program stored on the disk, together with how many bytes of memory each file takes up, and the amount of storage space left on the disk.

As with most disk systems, you can RENAME a file already stored on disk. Enter:

```
RENAME"OLDNAME.BAS" TO
   "NEWNAME.BAS"
```

This command renames the BASIC program OLDNAME as NEWNAME.

As well as storing files, or programs, on disk, you can also erase them from the disk. This is a useful command, as, unlike on a tape recorder, you cannot store a new program on an old one—but you can erase the old one and then store the new file in its place.

To erase a file named USELESS.BAS you would enter this:

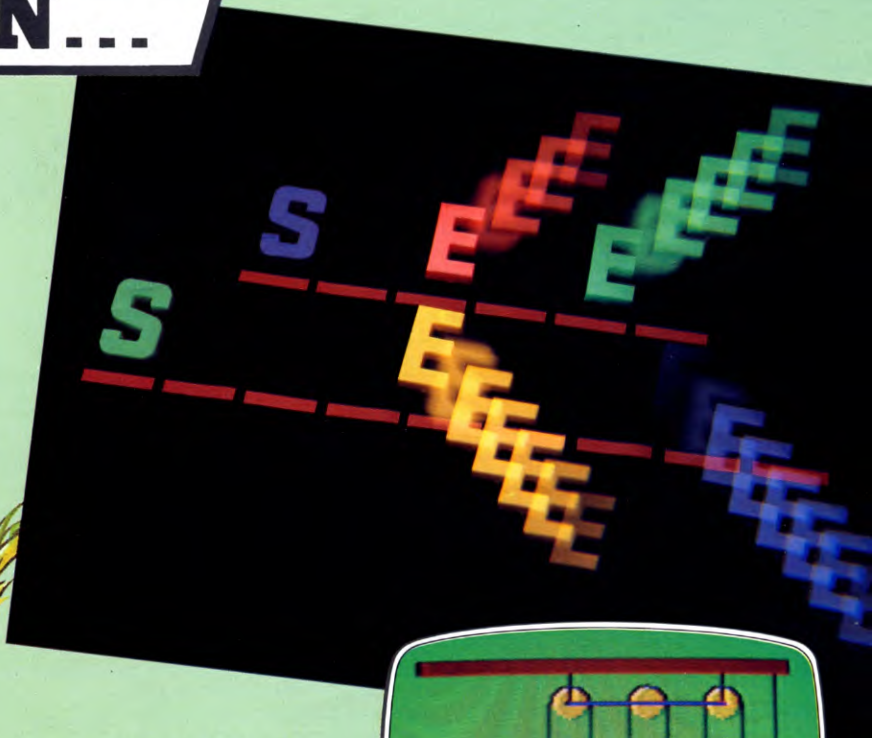
```
KILL"USELESS.BAS"
```

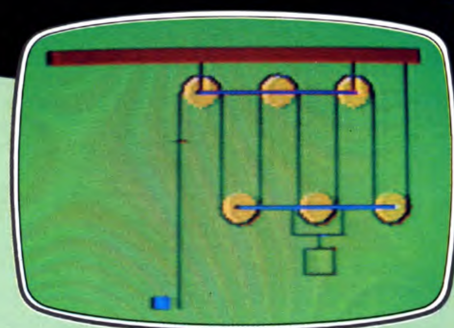
This rather dramatic-sounding command simply deletes the file you tell it to delete in the quotes.

VOLUME 3 OF INPUT

COMING SOON...











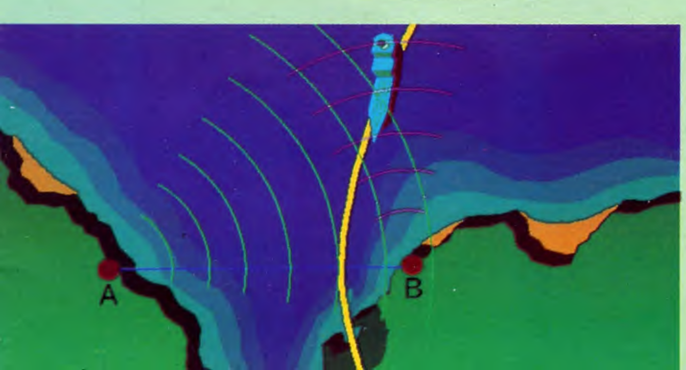
At last you can exploit your MACHINE CODE skills to the full: an exciting game—CLIFFHANGER—gives you arcade thrills and is a real challenge to your game's abilities. But it is much more than that, it contains a library of routines that will help you in your own game construction. Also in MACHINE CODE amongst other articles is a neat INTERRUPT CLOCK that runs simultaneously with BASIC programs.

Continue to build up your practice and knowledge of BASIC. Programs include an examination of MECHANICS and a look at the practicalities of CONIC SECTIONS; also included, among others, are techniques for SPEEDING UP BASIC.

In GAMES there is a BUSINESS STRATEGY game, a tricky WORD GAME, a SUPERFRUIT gambling game—see if you win or lose—plus others.

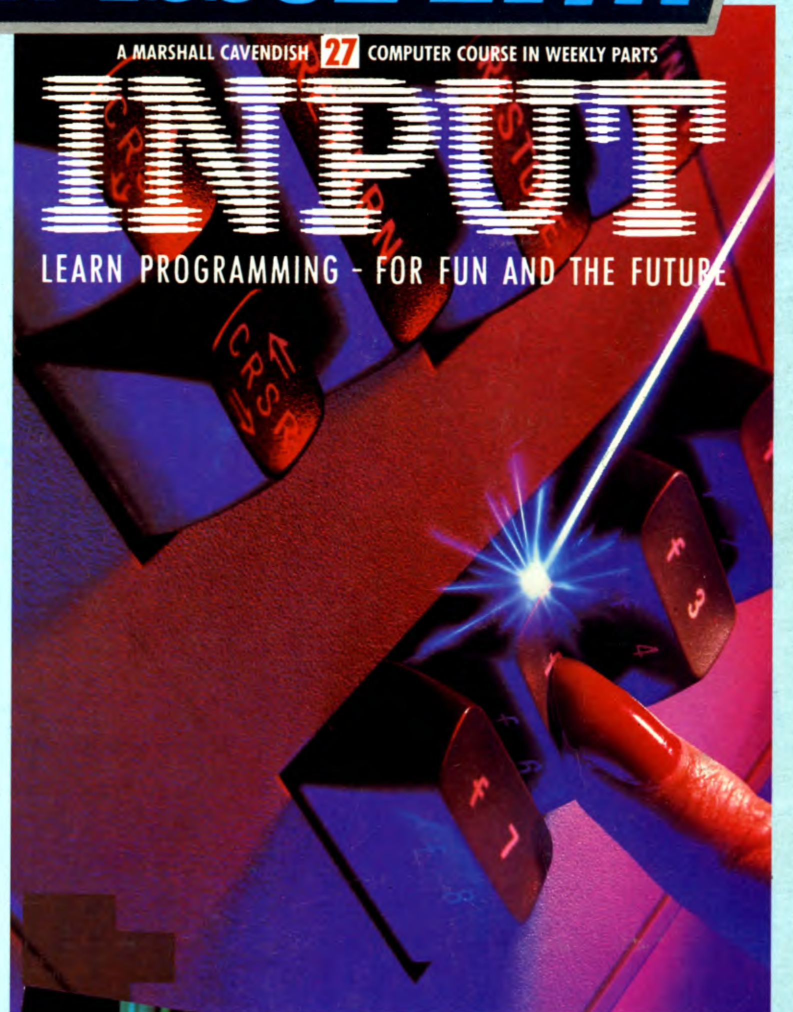
More APPLICATIONS for your Micro include an exploration of the possibilities of CODEWRITER programs; a ROOM DESIGN PLANNER; and a SOUND ANALYSER to digitalize recordings. Also a handy CALENDAR GENERATOR allows you to see at-a-glance how to plan your time throughout the year.

ROBOTS feature in PERIPHERALS—just how can you control them?

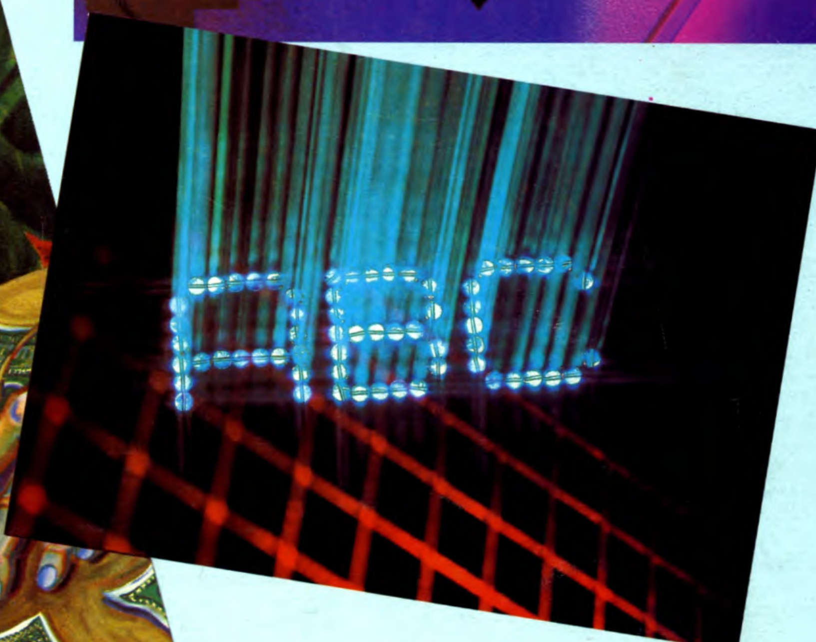


COMING IN ISSUE 27...

- *When in ROM ... well not always, especially if you ADD INSTRUCTIONS to make your micro respond to your very own BASIC*
- *Tired of your terrible typing? Take out the tears with a TEXT EDITOR program*
- *Keep on making it big with the second part of DISPLAY TYPEFACES*
- *Join the uptown gold diggers and find a viable vein in the first part of a GOLD MINING business strategy game*
- *Acorn and Commodore users! Find out about FUNCTION KEYS and save time with some special-purpose programming*







ASK YOUR NEWSAGENT FOR INPUT